AFIT/GE/ENG/92D-13

SATCOM General Purpose Modem
DSCS III SCT
Beacon Telemetry Display

THESIS

James D. Coppola
Captain, USAF

AFIT/GE/ENG/92D-13

AFIT/GE/ENG/92D-13

SATCOM General Purpose Modem

DSCS III SCT

Beacon Telemetry Display

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

James D. Coppola, B.S.E.E.

Captain, USAF

December, 1992

Acknowledgments

As with any large endeavor, there are a number of people who made significant contributions to ensure success. This thesis in certainly no exception.

First and foremost, I would like to thank my wife Robin, and my children MaryJane, Nicole, Erin, and Jacob for their help, indulgence, and patience over the past 18 months.

Thanks also to my thesis advisor, Major Mark Mehalic. He always had time for questions and kept me on track, especially with the documentation work!

Finally, I'd like to express my sincere appreciation to the Satellite Communications Group in the Wright Laboratory who let me use their equipment, facilities, and expert advice. Special thanks to Allen Johnson, Jim Foshee, and John Woods; Raytheon contractors Claude Begin, Jim Sheets, and Bill Brown; as well as TSSI contractors Tim Cellars and Ernie Barnhart. I hope the end product is something you can use!

James D. Coppola

Table of Contents

Page

List of Figures

List of Tables

Abstract

This thesis documents the design and implementation of a DSCS III Single Channel Transponder (SCT) beacon telemetry display. The system is a personal computer based design which interfaces to three SCT beacon receiver/demodulators. The software was designed to decode and display both the DSCS III A and DSCS III B satellite beacons. Recordings of the SCT beacon display can be made on paper and/or magnetic media when triggered by the user, a watchdog timer, or the SCT command accept telemetry bit. In addition, the system can be configured with an IRIG B Universal Time Coordinates (UTC) card which enables the software to determine the difference between the decoded SCT clock time and the local IRIG time source. Remoting the SCT configuration display is also possible using Hayes-compatible modems over a telephone link.

SATCOM General Purpose Modem

DSCS III SCT

Beacon Telemetry Display

I. Introduction

1.1 Background

1.1.1 General Purpose Modem The Satellite Communications (SATCOM) Group in the Avionics Directorate of Wright Laboratory provides satellite communications test support to the Department of Defense. As part of a continuing effort to upgrade support capabilities, the Group procured a general purpose modem (GPM) from ComQuest Technologies. This modem was designed to operate at several user-selected data rates and phase-shift-keyed (PSK) modulations. It is also capable of being remotely controlled by a separate host computer. The capabilities of interest were the 800 bits-per-second (BPS) receive only binary PSK (BPSK) mode, and the remote control function.

The 800 BPS receive only mode was specifically installed to demodulate the Defense Satellite Communications System (DSCS) III Super High Frequency (SHF) telemetry beacon. Included in the telemetry data is the configuration of the Single Channel Transponder (SCT) which is a package hosted on the DSCS III satellite. Additional background on the SCT is provided below. The GPM was designed to demodulate the beacon, decode the SCT configuration telemetry, and present the data to an external host. The host is responsible for interpreting and displaying the configuration data.

1.1.2 DSCS III SCT The Single Channel Transponder is a package attached to the host DSCS III satellite. The primary mission of the SCT is to disseminate Emergency Action Messages

(EAMs) to the nuclear capable forces [1]. The DSCS III primary communications subsystem includes eight antennas which can be interfaced in different ways to six transponders [2]. The SCT can be configured to use one of three uplink antennas; an earth coverage SHF, a multi-beam SHF, or an earth coverage Ultra High Frequency (UHF) antenna. On the downlink, the SCT uses an earth coverage UHF antenna on the DSCS III A satellites. DSCS III B satellites also have access to the channel 1 SHF transponder. In addition to frequency band choices, the SCT can be configured to use different modulations and cryptographic devices [3]. These various configuration selections make the SCT quite versatile. However, what makes it truly unique is the configuration changes can be implemented by a user terminal operating on the communications channel [4]. This is unusual because most satellite configuration changes are made by a ground controller through a dedicated channel called the Telemetry, Tracking and Control (TT&C) channel.

The SCT flexibility comes with a cost of increased complexity, which can cause satellite access problems. For example, if an airborne terminal is attempting to turn on the SCT downlink using the UHF channel while the SCT is configured for SHF, the attempt will fail. To avoid this problem, users may call a DSCS Operations Center to determine the SCT configuration, or request a configuration change to the desired mode. Obviously, it may be difficult for an airborne terminal to make a phone call. To solve having to go up "blind" (not knowing the SCT configuration), some terminals are equipped with a DSCS III SHF telemetry beacon receiver which can extract the SCT configuration information from the beacon. Once the configuration is known, the terminal can be modified to properly access the SCT.

One function of the ComQuest GPM is the demodulation and decoding of the DSCS III beacon telemetry. The SCT configuration information consists of 100 bits embedded in the beacon signal. The GPM was designed to present the bits of beacon data to an external device. The major thrust of this thesis was to obtain the data and display the decoded information in a user-friendly manner.

The using terminal must not only match the SCT configuration, its clock must also be synchronized with the SCT clock. Included in the SCT telemetry is the time value of the SCT clock [5].

Some terminals use the decoded beacon time to set their local terminal clock to quickly access the SCT. Other terminals do not have this capability, so they must depend on the accuracy of their local time standard and the SCT on-board clock. The SCT terminal owned by the SATCOM Group, the Command Post Modem/Processor (CPM/P), falls into the latter group of terminals. It cannot set system time from the beacon time directly. In order to reduce SCT access time for this terminal, this thesis compared the decoded beacon time with the local time standard to determine the difference.

1.2 Problem Definition

The SATCOM Group of Wright Laboratory procured a general purpose modem which demodulates and decodes the DSCS III SCT telemetry data. In order to be fully utilized, however, some method of capturing and displaying the telemetry data was required. In addition, the difference between the SCT clock and Universal Time Coordinates (UTC) needed to be determined to reduce satellite access time.

1.3 Approach

A literature review was accomplished to determine if any previously published work existed describing beacon telemetry displays. The research was expanded to include literature on automated testing to gain background information for display and test systems. Once the literature search was completed, the interface work with the GPM was started.

The SCT telemetry display entailed interfacing the GPM to a radio frequency (RF) front end, receiving the demodulated and decoded data from the modem, and displaying the data in a user-friendly form. The RF front end consisted of a SHF antenna, a low noise amplifier (LNA), and a

downconverter. The telemetry display platform chosen was an IBM compatible personal computer (PC). The software to control the PC was developed using Borland's TURBO C++ Professional software development system.

In addition to the SCT telemetry display of the GPM data, the software was also interfaced to two other SATCOM Group assets which provide SCT telemetry data. These assets were the Raytheon Stand Alone Beacon Receiver (SABR) and the AN/ASC-30 Small EHF/SHF terminal. Both these assets required additional interface software, while the AN/ASC-30 terminal required additional hardware.

1.4 Current Literature

The intention of this section was to survey the topics of satellite beacon telemetry displays and automated testing. While documentation on the general topics of satellite beacons and automated testing was abundant, information about the specific topics of this thesis was sparse. In doing the research, however, it was discovered the current literature provided information on topics similar to the ones presented in this report. As a result, this chapter's emphasis was changed to surveying the current literature on comparable approaches to solving telemetry data displays and implementing an automated test capability.

1.4.1 DSCS III SCT The only source found which discussed SCT displays was a technical memorandum by Laux of Wright Laboratory [6]. The report described a Z-80 microprocessor-based system which received IEEE-488 bus data packets provided by a SCT terminal and displayed the information using a custom LED display. Laux also mentioned a follow-on project which would bypass the SCT terminal and input the 100 bits of data directly, but no documentation was available on this project. In a related area, Robinson presented a method by which the Inter-Range Instrumentation Group (IRIG) time code standard was generated using bus-level time code translators [7] . The author discussed the advantages of using the newer bus level interfaces rather

than the traditional external interfaces. Two examples were presented, one using the PC AT/XT bus which was the bus used for this research.

1.4.2 Automated Testing This area of the literature survey was used to identify if a personal computer could be used as the display platform for the beacon display software. Several articles describing automated testing and testing using a personal computer were discovered. Lacy discussed microwave instrumentation systems enhanced by microprocessors in his conference presentation [8]. Of special interest was his discussion on how to use a personal computer as a platform for automated measurements. Specifically, he examined what factors to consider when configuring a PC for use in test and measurement applications, as well as what tradeoffs are possible. Chimene also discussed architectures, but did not limit his data collection platform to the PC [9]. He presented a technique which accepts data from subsystems at various clock rates and times. His technique separates the data collection platform from the telemetry system clock rate. Simms and Butterfield presented a telemetry system which not only captures data, but also formats and organizes it for future use [10]. The paper discussed the PC hardware required for the system. A related paper discussed software developed to process data in real-time and display a "quick look report" for display. Zimmerman presented a paper which contained structured BASIC software tools, utilities, disc input/output, and integrated measurement [11]. The tool kit provides a means for integrating routines for automated test systems. With the literature review completed, work was started on providing a SCT beacon display for the GPM.

1.5 Overview

Chapter II is an functional description of the system. It describes the hardware interfaces, cabling requirements, and the Turbo C programs. Chapter III presents an operational description of the software developed for the SCT telemetry display. The chapter is essentially an extended users guide which walks through each menu selection describing what functions are available to

the user. Chapter IV provides start-up procedures which details step-by-step instructions on how to bring up the telemetry display system and configure it for desired operation. Chapter V ends the thesis with some conclusions and recommendations. Appendix A provides a list of acronyms while Appendix B contains the source code listings for all the C code used in the telemetry display system.

II. Functional Description

2.1 Introduction

This chapter provides a functional description of the SCT beacon display system. The chapter starts with a background section on the DSCS III SHF beacon, then follows with sections detailing both the hardware and software developed. The hardware section deals with the interfaces the software expects to use. The software discussion describes the main features of the source code and what functions are in each file.

2.2 DSCS III SHF Beacon

The SCT configuration information consists of 100 bits. The bit definitions are shown in Tables 1,2, 3, 4, and 5. Table 1 define the clock bits for both A and B model SCTs. Tables 2 and 3 detail the rest of the telemetry bits for DSCS III A satellites, while Tables 4 and 5 show the definitions for DSCS III B satellites.

The 100 bits are embedded in a 800 bits-per-second (BPS) pseudo-random noise (PRN) data stream. The PRN stream is used to modulate the SHF binary phase shift Key (BPSK) beacon signal. It is broken up into 8-bit frames with each frame 10 milliseconds (ms) long. Table 6 shows how the frame is partitioned. The first bit is always a logic 0 and is called the sync bit. This allows the demodulator/decoder to frame synchronize to the data stream. Bits 1 through 5 are outputs of feedback shift-register sequence generators. The sequences are 25, 27, 29, 31, and 32 bits long. Bit 6 is a modulo-2 (binary) add of a logic 1 with bit 4 (31-bit sequence). Bit 7 is a modulo-2 add of the SCT telemetry bit with bit 2, the 27-bit sequence.

The SCT telemetry generator block diagram is shown in Figure 1 [3]. The telemetry information is updated every two seconds. It is then shifted out at 50 BPS into a $\frac{1}{2}$-rate convolutional encoder. The encoder adds in a bit for every beacon data bit, thus arriving at the 100 BPS rate which is used by the PRN generator in generating the 800 BPS modulating signal. It should be

Table 1. DSCS III Telemetry Clock Bit Definitions

Bit	Definition	Bit	Definition
1	Clock Days 40	28	Clock Milliseconds 800
2	Clock Days 20	29	Clock Milliseconds 400
3	Clock Days 10	30	Clock Milliseconds 200
4	Clock Days 08	31	Clock Milliseconds 100
5	Clock Days 04	32	Clock Milliseconds 80
6	Clock Days 02	33	Clock Milliseconds 40
7	Clock Days 01	34	Clock Milliseconds 20
8	Clock Hours 20	35	Clock Milliseconds 10
9	Clock Hours 10	36	Clock Milliseconds 05
10	Clock Hours 08	37	Clock Microseconds 4000
11	Clock Hours 04	38	Clock Microseconds 2000
12	Clock Hours 02	39	Clock Microseconds 1000
13	Clock Hours 01	40	Clock Microseconds 500
14	Clock Minutes 40	41	Clock Microseconds 400
15	Clock Minutes 20	42	Clock Microseconds 200
16	Clock Minutes 10	43	Clock Microseconds 100
17	Clock Minutes 08	44	Clock Microseconds 50
18	Clock Minutes 04	45	Clock Microseconds 40
19	Clock Minutes 02	46	Clock Microseconds 20
20	Clock Minutes 01	47	Clock Microseconds 10
21	Clock Seconds 40	48	Clock Microseconds 05
22	Clock Seconds 20		
23	Clock Seconds 10		
24	Clock Seconds 08		
25	Clock Seconds 04		
26	Clock Seconds 02		
27	Clock Seconds 01		

Table 2. DSCS III A Telemetry Bit Definitions

Bit	Definition
49	Broadside Command Bit 18
50	Broadside Command Bit 19
51	Broadside Command Bit 20
52	Bit-Error-Rate Enable(1)/Disable(0)
53	Command Address Enable(1)/Disable(0)
54	Uplink Frequency Enable(1)/Disable(0)
55	LSG WOD 1 Enable(1)/Disable(0)
56	LSG WOD 2 Enable(1)/Disable(0)
57	Uplink Modulation AFSATCOM II(1)/AFSATCOM I(0)
58	Downlink Modulation AFSATCOM I(1)/AFSATCOM II(0)
59	Uplink Bandwidth Narrow(1)/Wide(0)
60	Downlink Bandwidth Wide(1)/Narrow(0)
61	Uplink Signal Strength Bit 0 (LSB)
62	Uplink Signal Strength Bit 1
63	Uplink Signal Strength Bit 2
64	Uplink Signal Strength Bit 3 (MSB)
65	Decovered Baseband Bit 0 (LSB)
66	Decovered Baseband Bit 1
67	Decovered Baseband Bit 2 (MSB)
68	AFSATCOM I Baseband Mark(1)/Space(0)
69	Receiver Mode Bit 0 (LSB) : 00=SHF Only/01=SHF Only
70	Receiver Mode Bit 1 (MSB) : 10=UHF Only/11=Commutate
71	STG Update Enable(1)/Disable(0)
72	SHF Antenna Earth Coverage(1)/MBA(0)
73	Command Accepted Accepted(1)/Reset(0)
74	AFSATCOM II Downlink Hopping(1)/Fixed(0)

Table 3. DSCS III A Telemetry Bit Definitions (Continued)

Bit	Definition
75	AFSATCOM II Uplink Hopping(1)/Fixed(0)
76	Serial Magnitude Command bit 7
77	Serial Magnitude Command bit 8
78	Serial Magnitude Command bit 9
79	Uplink Crypto ILSG(1)/KI-35(0)
80	Downlink Crypto LSG(1)/KI-35(0)
81	Decover Enable(1)/Disable(0)
82	AFSATCOM I Bypass Enable(1)/Disable(0)
83	Downlink Bypass(1)/Standby(0)
84	Downlink EAM(1)/Standby(0)
85	SHF Bit-Error-Rate Count bit 0 (LSB)
86	SHF Bit-Error-Rate Count bit 1
87	SHF Bit-Error-Rate Count bit 2
88	SHF Bit-Error-Rate Count bit 3
89	SHF Bit-Error-Rate Count bit 4
90	SHF Bit-Error-Rate Count bit 5
91	SHF Bit-Error-Rate Count bit 6
92	SHF Bit-Error-Rate Count bit 7 (MSB)
93	UHF Bit-Error-Rate Count bit 0 (LSB)
94	UHF Bit-Error-Rate Count bit 1
95	UHF Bit-Error-Rate Count bit 2
96	UHF Bit-Error-Rate Count bit 3
97	UHF Bit-Error-Rate Count bit 4
98	UHF Bit-Error-Rate Count bit 5
99	UHF Bit-Error-Rate Count bit 7 (MSB)
100	Classified Telemetry Enable(1)/Disable(0)

Table 4. DSCS III B Telemetry Bit Definitions

Bit	Definition
49	Broadside Command Bit 18
50	Broadside Command Bit 19
51	Broadside Command Bit 20
52	Bit-Error-Rate Enable(1)/Disable(0)
53	Command Address Enable(1)/Disable(0)
54	Uplink Frequency Enable(1)/Disable(0)
55	LSG WOD 1 Enable(1)/Disable(0)
56	LSG WOD 2 Enable(1)/Disable(0)
57	Uplink Modulation AFSATCOM II(1)/AFSATCOM I(0)
58	Downlink Modulation AFSATCOM I(1)/AFSATCOM II(0)
59	Uplink Bandwidth Narrow(1)/Wide(0)
60	Downlink Bandwidth Wide(1)/Narrow(0)
61	Uplink Signal Strength Bit 0 (LSB)
62	Uplink Signal Strength Bit 1
63	Uplink Signal Strength Bit 2
64	Uplink Signal Strength Bit 3 (MSB)
65	Decovered Baseband Bit 0 (LSB)
66	Decovered Baseband Bit 1
67	Decovered Baseband Bit 2 (MSB)
61	Continuous SHF Downlink Pwr Level Bit 0 (LSB)
62	Continuous SHF Downlink Pwr Level Bit 1
63	Continuous SHF Downlink Pwr Level Bit 2
64	Continuous SHF Downlink Pwr Level Bit 3
65	Continuous SHF Downlink Pwr Level Bit 4
66	Continuous SHF Downlink Pwr Level Bit 5
67	SHF Continuous Downlink Enable(1)/Disable(0)
68	VPC TLM (1) Bits 61-66,67/(0) 61-64,65-67 Defined
69	Receiver Mode Bit 0 (LSB) : 00=SHF Only/01=SHF Only
70	Receiver Mode Bit 1 (MSB) : 10=UHF Only/11=Commutate

Table 5. DSCS III B Telemetry Bit Definitions (Continued)

Bit	Definition
71	STG Update Enable(1)/Disable(0)
72	SHF Antenna Earth Coverage(1)/MBA(0)
73	Command Accepted Accepted(1)/Reset(0)
74	AFSATCOM II Downlink Hopping(1)/Fixed(0)
75	AFSATCOM II Uplink Hopping(1)/Fixed(0)
76	Serial Magnitude Command bit 7
77	Serial Magnitude Command bit 8
78	Serial Magnitude Command bit 9
79	Uplink Crypto ILSG(1)/KI-35(0)
80	Downlink Crypto LSG(1)/KI-35(0)
81	Decover Enable(1)/Disable(0)
82	AFSATCOM I Bypass Enable(1)/Disable(0)
83	Downlink Bypass(1)/Standby(0)
84	Downlink EAM(1)/Standby(0) (Both 83,84=Cont SHF)
85	SHF Bit-Error-Rate Count bit 0 (LSB)
86	SHF Bit-Error-Rate Count bit 1
87	SHF Bit-Error-Rate Count bit 2
88	SHF Bit-Error-Rate Count bit 3
89	SHF Bit-Error-Rate Count bit 4
90	SHF Bit-Error-Rate Count bit 5
91	UHF Downlink Enable(1)/Disable(0)
92	SHF Downlink Enable(1)/Disable(0)
93	UHF or SHF Bit-Error-Rate Count bit 0 (LSB)
94	UHF or SHF Bit-Error-Rate Count bit 1
95	UHF or SHF Bit-Error-Rate Count bit 2
96	UHF or SHF Bit-Error-Rate Count bit 3
97	UHF or SHF Bit-Error-Rate Count bit 4
98	UHF or SHF Bit-Error-Rate Count bit 5
99	UHF or SHF Bit-Error-Rate Count bit 7 (MSB)
100	Classified Telemetry Enable(1)/Disable(0)

Table 6. Frame Bit Definitions

Bit	Name	Sequence
0	Sync	Logical 0
1	25 Bit	1000010011111000110111010
2	27 Bit	100001001011001111100011010
3	29 Bit	10000100101100111110001101110
4	31 Bit	1000010010110011111000110111010
5	32 Bit	10000010010110011111000110111010
6		Modulo-2 sum of logical 1 with bit 4
7		Modulo-2 sum of beacon data with bit 2

noted that the GPM is the only demodulator/decoder which takes advantage of the convolutional encoder by using it in conjunction with a Viterbi decoder to improve the bit-error-rate performance. The other two receivers just ignore the extra bit.

In addition to the frame synchronization, the data stream provides a means to message synchronize. The initial message synchronization occurs when the 25-bit and 32-bit sequences are both a logic 1 for five frames. In other words, a decoder must monitor both sequences, and when they have reached the 11111 state for the past five frames, message synchronization has been found. The first bit of the SCT telemetry data (bit 1) is present when the message synchronization pulse occurs. The message synchronization pulse is present every eighth repeat of the 11111 state of the 25-bit sequence. Once message synchronization has occurred, the beacon telemetry information can be obtained for display.

2.3 Hardware

This section describes the PC cards which were used to enhance system capabilities and defines the general hardware interface requirements for the beacon display system software.

2.3.1 PC Cards The PC standard configuration is usually one RS-232 communications port and one printer port. The system developed for this thesis added an additional serial RS-232 port (COM2), a parallel data input/output (PIO) card, and an Inter-Range Instrumentation Group

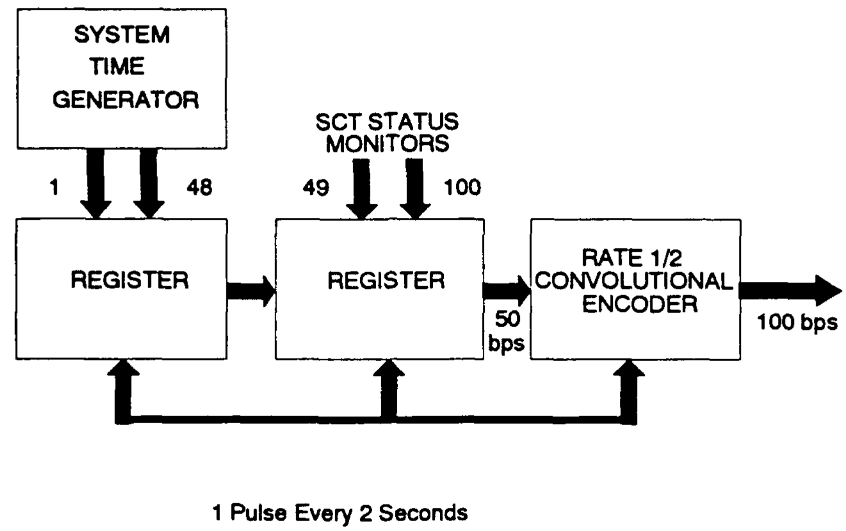

Figure 1. SCT Telemetry Generator Block Diagram

(IRIG) time standard card. Table 7 show the interrupt definitions, port addresses, and interrupt vectors used by the software.

Table 7. PC Interface Card Summary

Card	IRQ	I/0 Address	Interrupt	Vector
Com1	IRQ4	0x3F8-3FF	0x0C	0x30-33
Com2	IRQ3	0x2F8-2FF	0x0B	0x2C-2F
PIO	IRQ3	0x2F8-2FF	0x0B	0x2C-2F
LPT1	IRQ7	0x370-37F	0x0F	0x3C-3F
IRIG	IRQ5	0x320-32F	0x0D	0x34-37

The parallel card is a 24-bit parallel input/output device developed by Industrial Computer Source [12]. The card plugs into a PC-XT slot and comes with an external 37-pin D connector. The input levels for the card are transistor-transistor-logic (TTL) levels (0,+5V).

The IRIG card was developed by Bancomm Corporation [13]. The card can be synchronized to an external IRIG-B source and provide scheduled interrupts to the PC. The device is installed in a PC-AT slot and is interfaced to an external IRIG standard through a BNC connector. The

display system software programs the card to synchronize to the external IRIG-B time code source and provide a 1 pulse-per-second (PPS) on-time interrupt for system time.

The other devices used by the system are the COM1 and COM2 RS-232 serial ports and the parallel LPT1 port for the printer. The COM1 and COM2 ports are used by the software to interface to the beacon receivers. Table 8 list the PC port usage by the system.

Table 8. PC Port Summary

Port	GPM	Raytheon	ASC-30
Com1	J6A Beacon	J3 Beacon/Control	Not Used
Com2	J1 Control	Not Used	Not Used
PIO	Not Used	Not Used	Beacon
LPT1	Printer	Printer	Printer

2.3.2 Beacon Receiver Interface

2.3.2.1 GPM The hardware interface to the GPM is through the PC RS-232 serial communications ports. For the GPM, both COM1 and COM2 are used. Table 9 shows the interface settings used. COM1 is used to read in the beacon data from the GPM. It must be connected to the J6A beacon connector on the back of the GPM through a ComQuest provided 20-to-9 pin adapter cable. COM2 is used to communicate with the GPM through its remote host port. The control capability is provided by the system software. The port settings shown in Table 9 are the factory remote-host settings for the GPM [14]. The remote-host connector, J1, is also located at the back of the GPM.

Table 9. GPM Interface Settings

Port	Baud	Data	Parity	Stop
Com1	9600	7 Bit	Odd	0 Bits
Com2	1200	7 Bit	Odd	0 Bits

It should be noted that the remote software also uses COM2 when remoting the display to another site. Therefore, when the GPM is used as the beacon receiver, the control functions are disabled when the display is remoted by starting the remote master software.

2.3.2.2 Raytheon The interface to the Raytheon beacon receiver is through COM1 only. The port settings are shown in Table 10. The COM1 port of the PC must be cabled to the receiver host port, connector J3, on the back of the system. COM1 is used to send start-up and control data to the receiver, as well as receiving the demodulated and decoded data from the receiver.

Table 10. Raytheon Interface Settings

Port	Baud	Data	Parity	Stop
Com1	9600	8 Bit	None	1 Bit

2.3.2.3 ASC-30 The ASC-30 interface is more complicated than the others. The ASC-30 sends a synchronous signal at 800 BPS. It provides demodulated data only, not demodulated and decoded data as the other receivers. In order to get synchronous data into the PC, the DIO-24 PIO card was used. The telemetry display system software configures the card for interrupt operation and data input mode. The pins used are shown in Table 11. Note that pins 1 and 20 are tied together to enable interrupts [12].

The DIO-24 requires TTL voltage levels (0, +5V). The ASC-30 data and clock signals are at RS-232 levels (+12, -12V). An MC1489 line receiver was used to convert the signals to the TTL level required. The MC1489 was installed in the SABR DAC interface system developed from prior beacon work [1].

Table 11. DIO-24 Pins

DIO-24 Pin	Signal Name
37	800 BPS Data
1	Beacon Clock
19	Ground
1 wired to 20	INT Enable Jumper

2.3.2.4 Remote Equipment The SCT beacon display system can be remoted to another location using Hayes-compatible modems. The remote software can be used only with the GPM and Raytheon receivers. In addition, the remote master software uses COM2 to relay the

data, so the GPM control functions are disabled when remoting the display. The communications settings are shown in Table 12. The modems used were ZOOM model FX9624 and CTS DATACOM model 2224ADH. It is recommended the DATACOM or a ZOOM modem be used at the local site, while the ZOOM be used at the remote site. Erratic behavior occurs when the ZOOM tries to initiate the connection to the DATACOM. The receive modem detects the ring, but when it puts up the carrier both modems disconnect.

Table 12. Remote Interface Settings

Function	Port	Baud	Data	Parity	Stop
Master	Com2	2400	8 Bit	None	1 Bit
Slave	Com1	2400	8 Bit	None	1 Bit

2.4 Software

This section details the software developed for the beacon display system. Each C language file is discussed and salient features of particular functions in the files are presented.

2.4.1 Overview The software was developed using the Borland Turbo C++ Professional software development system. The Turbo-C project facility with the interactive editor environment was used to create the system. The project facility allows multiple file programs and provides all the MAKE dependency information to compile and link the entire program.

The project consists of 12 header files (.H files) and 19 C files (.C files). The project contains approximately 16,000 lines of compiled code which links to a 100,000+ byte executable file. The executable is called "SCT.EXE". Tables 13 and 14 list the files and their sizes.

2.4.2 Design The initial design methodology tried was an object oriented approach. After the first design review, however, the actual system design was developed using a functional design methodology. The reason for the change was limited experience with the object oriented paradigm. The final product was divided into six functional blocks. The blocks are: receiver, decode and

Table 13. .C Files

Name	Size
ASC30.C	7191
CONTROL.C	6231
CURSOR.C	3068
GPM.C	3303
INIT.C	1983
IRIG.C	6868
KEYBOARD.C	2753
MAIN.C	5448
MENU.C	4166
OPTIONS.C	12969
PIO.C	2769
PORT.C	7936
PRINTER.C	1632
RAY.C	2884
REMOTE.C	6165
SCREEN.C	17640
STIME.C	2797
TIMER.C	2465
WINDOW.C	5390

Table 14. .H Files

Name	Size
CURSOR.H	462
INTERUPT.H	681
IRIG.H	1367
KEYS.H	1624
MENU.H	1200
MODEM.H	412
PIO.H	1187
PORT.H	1731
SCREEN.H	357
SYSTEM.H	1215
TIMER.H	558
WINDOW.H	645

display, system time, user interface, remote, and control. Each block may contain additional submodules which were required to implement software primitives for the parent functional block. In addition, the SCT telemetry display system was designed as a software state machine. The states were aligned with the functional blocks. The primary functions of each block and the states of the system are discussed below.

2.4.2.1 Receiver Block The receiver functional block provides the interface to the three beacon receivers. All receivers required a unique software interface. The primary function of the receiver software was to obtain the 100 bits of beacon telemetry data from the receiver, and then call the decode and display functional block to display the information. The GPM and Raytheon interfaces use the interrupt driven RS-232 port software to capture the input data from the receiver. Interrupt driven port software was required because the beacon receivers provide data at 9600 baud which is too fast for polled port software to operate. The ASC-30 receiver uses the PIO interface (also interrupt driven) to capture the 800 BPS data delivered by the AN/ASC-30 beacon receiver. Additional functions were required in this interface to locate the frame sync bit and obtain message synchronization. Once message synchronization is found, the frame sync bit is monitored to ensure the software stays synchronized to the 800 BPS PRN sequence.

All three receiver software modules load the captured beacon telemetry data into a global array named data. Once all bits have been obtained, routines in the decode and display functional block are called to update the PC screen display.

2.4.2.2 Decode and Display Block The decode and display functional block was designed to decode the 100 bits of SCT telemetry information delivered from the receiver functional block and display the information on the PC screen. The block also uses several global variables to determine which model satellite to display (DSCS III A or DSCS III B), and the status information listed on line 25 of the telemetry display. The software only updates the screen when beacon

information has actually changed. This screen update method was used to minimize CPU cycles for screen updates saving them for data acquisition and processing.

In addition to being called by the receiver software, routines in the decode and display block are called by the system time block to display the system clock and date.

2.4.2.3 System Time Block The system time functional block routines were used to generate the system time and date for the beacon display. There are two timers available. The first is an MS-DOS timer which uses the BIOS timer function to generate a 1-PPS interrupt. When the interrupt occurs, a call to MS-DOS is made to get the PC time. The time value is then formatted and displayed on the screen. A call to the MS-DOS date software interrupt is made at timer initialization and when the hour is updated. The hourly call was installed to check for 24 hour roll-over. The timer interrupt occurs asynchronously which means that it may occur at any time.

The other timer available is an IRIG timer, provided a Bancomm IRIG card is installed in the PC. The software was designed to configure the IRIG card for a 1-PPS interrupt. When the interrupt occurs, an interrupt service routine requests time from the card, and then displays the values to the screen. The IRIG time format includes a date which is initially formatted and displayed to the screen, then monitored for changes. When a change occurs, the new date replaces the existing date on the PC screen. The IRIG timer interrupt can also occur at any time.

2.4.2.4 User Interface Block The user interface functional block was designed to provide the PC interface to the operator. It involves the keyboard and menu software drivers. The keyboard software chains to the BIOS keyboard interrupt service routine and intercepts all keyboard inputs when the beacon display is on the screen. When the menu software is employed, the keyboard interrupt is removed for menu processing. In addition, when the menu is displayed, the receiver software is disabled and telemetry updates are halted.

The menu software originates in the options section of the user interface block. The options software contains the menu data and the functions called when the user selects a menu item. It calls the menu software to display and manipulate the menus. The menu program in turn calls the window software to open and close windows for menu displays. This layered approach allowed the lower level software in the menu and windows programs to be designed with re-use in mind. Only the options software is specific to the SCT telemetry display system application.

2.4.2.5 Remote Block The function of the remote block is to remote the beacon display to another location. The design has two parts, one for the local display where the beacon receiver is located, and one for the remote location. The link currently supported by the software is over a phone line using Hayes-compatible modems (AT command set). The software at the local site is called the remote master and initiates the phone call. The software at the remote location is named the remote slave and must be up and waiting for the master to call for the system to synchronize properly. The remote slave answers the phone and initiates the handshaking to synchronize with the master. Once master/slave synchronization has been accomplished, the master sets a global flag informing the receiver block to also remote the incoming beacon data to the modem port, as well as send the satellite name at the end of the data stream. The remote slave software also sets a global variable for the receiver block enabling additional software which monitors the satellite name delivered by the remote master to update the PC screen when required.

2.4.2.6 Control Block The control functional block was designed to interface and command the GPM and Raytheon beacon receivers. This block was provided as a user enhancement so both of the receivers can be initialized and configured from within the SCT telemetry display program. This saves the user from having to exit and restart the program if a receiver configuration change is required.

2.4.2.7 State Diagram In addition to designing functional blocks for the SCT telemetry display, the program was designed to operate as a software state machine. Figure 2 shows the states defined and the transitions requirements. The syntax for the transition state diagram was developed by Rumbaugh [15]. The transitions into each state are shown on the arcs of the diagram. The DO functions are listed below and describe the primary functions of each state. If no transition condition is present on an arc, then the transition occurs when the DO function completes.

Menu DO operations:

1. Process User Selections.

Receiver DO operations:

1. Capture beacon data from beacon receiver or modem port, put in data array.
2. If remote master, relay incoming beacon data and satellite name.
3. If remote slave, perform 1 above then get satellite name.

Display DO operations:

1. Compare new data with old, update when bits are different.
2. If called from time, display new system time and date.

Timer DO operations:

1. 1 PPS interrupt occurred, update system time, update system date if required.

Remote DO operations:

1. Master enabled, configure modem, get phone number, call and synchronize with slave.
2. Remote enabled, configure modem, wait for master to call, answer phone and synchronize with master.

Control DO operations:

1. GPM Start enabled, send GPM beacon mode commands to GPM over COM2.
2. GPM Window enabled, send keyboard keystrokes to GPM, echo GPM responses to screen.
3. Ray Start enabled, wait for Raytheon to complete self test, get satellite from user, send proper oscillator commands to the Raytheon receiver.
4. Ray Offset enabled, get offset frequency from user, format and send oscillator commands to the Raytheon receiver.

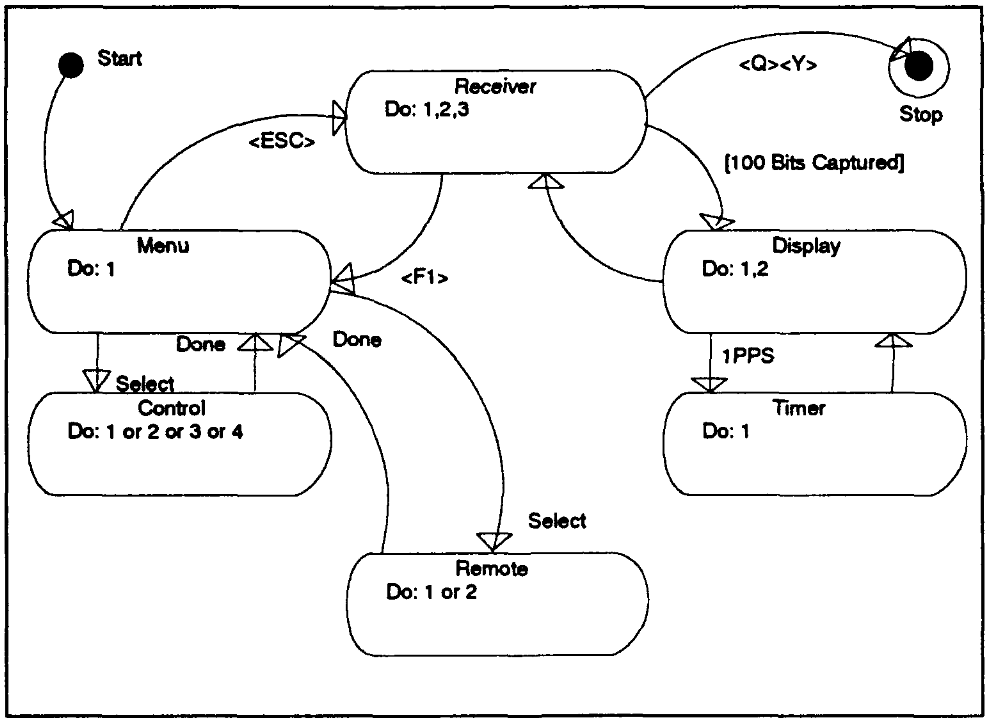

Figure 2. State Transition Diagram

2.4.3 Source Code The system development was an incremental process. It started with attempts to read data from the beacon receivers. The first interface attempted was to the ASC-30. The first interface completed was with the Raytheon system. Using the experience gained by

debugging the Raytheon software, the ASC-30 program was corrected and completed. The GPM interface was competed after the GPM was delivered. When it was delivered, however, it was discovered that the decoder did not operate properly. To help solve the problem, ASC-30 800 BPS data was captured and sent to ComQuest. The decoder problem was identified and the control ROMs were updated to correct the problem.

With the receiver interfaces completed, the telemetry decode and display software was designed and implemented for the DSCS III B satellite. After debug and test, the software was updated to include decoding and displaying of DSCS III A satellite telemetry.

Once the decode and display functions were complete, the keyboard interrupt drivers and initial menu software were developed. Changes to the menu software continued up through the end of the project.

With the delivery of the IRIG card, both system timer software functions were implemented. Calendar conversion routines were added to complete the beacon screen display. The system was then demonstrated to Wright Laboratory personnel and it was recommended that each satellite name be available for display. They also requested the remoting capability. Both additions were then implemented. The final enhancements (after the remoting capability) to the display system were the addition of the logging and control software, which enable users to document beacon configurations and control the GPM and Raytheon beacon receivers.

2.4.3.1 Start-UP and Control Files This section briefly describes the three files which contain the global variables, the control functions, and the initialization routines.

SYSTEM.H The SYSTEM.H file contains the system defined global variables. An attempt was made to limit the use of global variables, but timing and memory constraints forced their use to avoid parameter passing overhead. The global variables are actually declared in the MAIN.C file by the Turbo C compiler.

MAIN.C The required main() function resides in this file. The program entry and exit points occur here as well. The entry point is in main() where the optional command line argument is evaluated. Main() also calls init() to initialize system parameters, and then calls the selected receiver software for telemetry data capture. In addition, all menu and logging functions are called from main(). The MAIN.C file also contains the finish() routine which performs all necessary functions to gracefully exit the program to MS-DOS.

INIT.C This file contains functions which initialize system variables. The init() function also calls the options() function to display the menu for user initial inputs.

2.4.3.2 Receiver Files The files listed below contain the code that interfaces to the beacon receivers. The software uses the port driver software to control the data acquisition into the PC.

GPM.C The GPM.C file contains the routines necessary to decode the beacon telemetry bits from the ASCII characters sent by the GPM. The gpm() function opens the receiver port (COM1) for the data input. The GPM receiver packs the bits four at a time, from right to left (MSB -> LSB). Then, an offset is numerically added to create an ASCII character. The bits are all collected and packed, and then the characters are sent to the PC. At the end of the 100 bits of data, the GPM sends an "H" character. To start the next update, the receiver sends a carriage return and line-feed, followed by two spaces. The code traps on the line feed, ignores the spaces, then decodes the telemetry bits. Once the bits have been obtained, the display software is called to display the telemetry update.

There is additional software in GPM.C that detects when the GPM has stopped sending data. A notice is posted on the screen alerting the user. The notice is removed when data is received by the COM port.

RAY.C This file contains the functions required to obtain and parse the beacon telemetry bits from the characters sent by the Raytheon beacon receiver. The data format is different than

the GPM. The bits are grouped eight bits at a time, from left to right (LSB – > MSB), and no ASCII offset is added. In addition, the data stream is preceded by the string "START". The Raytheon receiver sends the telemetry bits as they are demodulated and packed, not waiting to collect all 100 bits. The software traps on the "START" string header, then strips out the data as it is sent. Once the 100 bits are obtained, the decode and display software is called to update the SCT configuration display. The Raytheon software also contains a port timer which notifies the user if the system is not receiving data from the beacon receiver.

ASC30.C The ASC-30 receiver software is more complex than the other two receiver packages. This is because the ASC-30 does not pull out the telemetry bits from the 800 BPS data stream. The ASC-30 is a demodulator only. The software first finds the sync bit in the data stream. Since the ASC-30 may phase-lock 180 degrees out of phase, the software searches for both a 1 and a 0 sync bit. An invert flag is set if data inversion is required.

Once the bit synchronization is found, the software attempts to find message synchronization by monitoring the 25- and 32-bit sequences for the 11111 pattern. When it is found, program control is passed to the telemetry recovery software where the beacon bits are decoded from the data stream. The sync bit is monitored for loss of synchronization.

The ASC-30.C software does not provide remoting capabilities. This is because the PIO card uses the COM2 interrupt request and vector, as well as requiring major software modifications to send the acquired data over the phone link. These changes were not implemented so other enhancements could be made in the time allowed.

Another problem with the ASC-30 receiver software is that it loses message synchronization when the display system is updating system time. This is due to the added computational load of the ASC-30 software. The system time functions are disabled when the ASC-30 program is called. They may be restored by using the menu system. The problem exists on 80286 (12 MHz or slower) CPU based machines.

2.4.3.3 Display Files The display files contain the code used to display the beacon data, configure and control the screen, and implement the menuing system. These files are the largest programs in the display system.

SCREEN.C / SCREEN.H The SCREEN.C file contains all the decode and display software for the telemetry display screen. It is the largest file in the project. The routines are called from the receiver software when a screen update is required. The decode software compares the new beacon data with a copy of past beacon data and then updates any changed information. This means that screen updates occur only when beacon information has been changed. This method was used to minimize screen writes, saving CPU cycles for the decoding software.

The routines are partitioned in the same manner as the display screen. There are routines for displaying system time, beacon time, beacon status, and beacon configuration. Also included are routines to update the lower left and lower right blocks of the screen. In addition, there is a routine to update the labels and borders. All blocks are relative to base points which are declared as package global variables.

SCREEN.H contains the file function prototypes and package variables used in SCREEN.C. In general, the .H files were created for interface purposes to define the functions available to external programs, and to declare package global variables required by functions in the file. Global function declarations are performed in the C language by means of function prototyping. Function prototypes are declarations (usually in a .H file) of the function name, the number and type of parameters passed to the function, and the type of the function itself (the type of value returned by the function). This allows external programs to call and interface properly to the prototyped function.

CURSOR.C / CURSOR.H These routines perform cursor manipulations. They are modified versions of code presented in the Turbo C book by Stevens [16]. While many functions were included, the only ones used were the hide cursor and display cursor programs.

OPTIONS.C This software package is the second largest file in the project. It contains the code which defines the menus for the system. The functions in MENU.C actually implement and display the menu, but the software in OPTIONS.C contain the data structures for the menus, as well as the programs which are executed when the user selects a menu item.

MENU.C / MENU.H This file contains the functions which display and manipulate the menus. This software is called when the user presses the $< F1 >$ function key. It is also a modification of software from [16]. When called, the software displays the menu information passed to it. The software monitors the arrow keys, the carriage return key, and the escape key. Each direction key moves the highlighted area of the menu. Carriage return executes the selected function, while escape exits the menu software. The code uses window primitives in WINDOW.C to open and close windows for each menu.

WINDOW.C / WINDOW.H This code allows the user to open and close windows. The current active window is the last one opened. The software can support up to ten open windows. In addition to windowing functions, the software provides display notice, error message, and yes/no input specialty windows.

2.4.3.4 System Time This section of the system software generates the the system time for the display and the beacon clock offset. In addition, calendar functions are contained in the files to convert the dates for display to the PC screen.

STIME.C The STIME.C file contains the routines used to produce the MS-DOS system timer. This timer is implemented using the BIOS timer interrupt. A counter in the timer interrupt service routine determines when to update the system time.

In addition to the timing functions, calendar functions were used to change the MS-DOS date in terms of day and month to a linear day of year. January 1 is considered to be day 1. The calendar conversion routines are conversions of PASCAL public domain software found in [17].

IRIG.C / IRIG.H IRIG.C software disables the MS-DOS timer, initializes the IRIG card, and provides an interrupt service routine to update the system time and calculate the beacon clock difference. It also provides routines to disable the card and interrupt driver. The initialization routine configures the IRIG card to synchronize to an external IRIG B time standard, then interrupt the PC at one second intervals when the "ON TIME" IRIG event occurs.

The IRIG.H file contains the port address, interrupt request (IRQ), and interrupt vector assignments for the card. If the IRIG hardware address or interrupt values are changed, the values in this file must be updated to match the change, or the software will not be able to access the card.

There is an intermittent problem with the IRIG system timer. It usually occurs when the program is interrupted and the IRIG card is not reset or the interrupt not disabled. The card still initializes normally, but will not generate the 1 PPS interrupt. The fix was to turn off the PC for at least one minute. This resets the card for normal use. It is not know why the problem occurs, or why the fix works.

2.4.3.5 Ports The port files contain the software primitives to control the data acquisition ports. The ports used were the COM1 and COM2 serial RS-232 ports and a DI0-24 parallel port.

PORT.C / PORT.H PORT.C contains the software required to interface with the PC serial ports. These are the port drivers. The current version of the software supports the COM1 and COM2 ports. The receive software is interrupt driven and provides a buffer of 1024 characters for received data. The transmit routines are not interrupt driven, but will wait until the transmit Universal Asynchronous Receiver/Transmitter (UART) is ready so data is not overwritten. The code provides interrupt service routines for both ports. In addition, the receive software will relay data to the remote port (COM2) when the display system is in the remote mode.

The PORT.H file contains the function prototypes, but more importantly, contains the port addresses, IRQs, and interrupt vector definitions for the RS-232 ports. The values are shown in Table 7 used are the standard defined for the PC AT [18].

PIO.C / PIO.H These files contain the software to initialize and use the DIO-24 PIO card. Like the COM ports, the PIO software provides interrupt driven receive capability with a 1024 character buffer. The code also provide a transmit capability, but it is not used.

Like the PORT.H file, the PIO.H file contains the address, IRQ, and interrupt vector definitions for the DIO-24 card. Again, any changes to the hardware settings require changes in PIO.H so the software can communicate with the card.

2.4.3.6 Remote This section of the display system software was implemented at the request of the Wright Laboratory. The functions take the beacon display running at the receiver location and remote the display to a different site using a telephone link.

REMOTE.C REMOTE.C has four main routines which implement the start remote master, start remote slave, stop remote master, and stop remote slave functions. The start routines initiate the modem, dial or answer the phone, synchronize with each other, and then display the beacon telemetry data. The start remote master sets a global flag telling the receive port software to relay data to COM2 for remoting. The start remote slave software opens the COM1 port, receives the receiver type from the master, then exits to the proper receiver software.

Both stop routines hang up the phone and reset the modem. The remote master will detect when the remote slave hangs up and stop relaying the data. The remote slave will display a "NO DATA" notice if the master stops sending data.

MODEM.C / MODEM.H The MODEM.C software was incorporated into the remote file. It contained the primitives used to control and use the modems. The software assumes Hayes compatible modems using the AT command set.

MODEM.H contains the actual modem control strings. If different modems are required, then this is the file to modify with the proper modem control instructions.

2.4.3.7 Beacon Receiver Control The control file was added to allow the user to bring up and configure the GPM and the Raytheon receivers without exiting the SCT telemetry display program.

CONTROL.C The four receiver control files are located in CONTROL.C. The two GPM functions are gpm_start() and gpm_window(). The start code configures the GPM for beacon operation. The window software allows the user to send commands to the GPM. It takes the keyboard input and sends it to the GPM, and then echoes all GPM responses to the PC screen in the active window. The GPM window software was specifically installed to reset the GPM back to the factory default settings (the IN command) [14]. The GPM RS-232 port (connector J1) must be set to the factory default sections which are listed in the hardware section of this chapter.

The two Raytheon control programs are ray_start() and ray_offset(). Both programs are conversions of the Raytheon Pascal code delivered with the beacon receiver. The C equivalent programs can replace the PASCAL program if desired. The start code allows the user to bring up and configure the receiver. The offset function calculates frequency offset commands to be sent to the receiver. The frequency offset code was included and tested for completeness, but frequency offsets were never required during the work on this thesis.

2.4.3.8 Support Files The files listed below provide software primitives for control of the PC hardware.

TIMER.C / TIMER.H These files allow the system to chain to the BIOS system timer interrupt. The interrupt service routine is used to determine port timeouts and to implement the MS-DOS system time generator.

INTERUPT.H This file contains a general list of interrupt definitions used throughout the system. The file was developed with re-use in mind, so it contains several definitions that are not used.

KEYS.H This file contains the PC BIOS keyboard scan codes used by the system.

KEYBOARD.C Keyboard control routines are located in this file. It chains to the existing keyboard interrupt and provides an interrupt service routine to obtain keystrokes for the display system. It also contains a versatile get string routine (my_cgets) which is used for all user string inputs.

PRINTER.C As the file name suggests, this file has the printer driver routines in it. Included is a function checking the printer port for problems. It was developed to ensure the system did not crash when the printer had a problem.

2.5 Summary

This chapter presented an overview of the SCT telemetry data format, the hardware used by the display system, and the source code developed for the system. Chapter III presents an operational view of the SCT display system.

III. Operational Description

3.1 Introduction

This chapter provides an operational description of the SCT telemetry display software. It starts with an overview of how the system was designed to operate, follows with how to bring the system up, and then discusses the different operating modes available to the user. After covering system modes, each menu selection is detailed as to its function and what it does to the operation, or state, of the SCT telemetry display system.

3.2 Overview

The SCT telemetry display software was designed to run on an IBM compatible personal computer and interface to one of three DSCS III SHF beacon receivers. The receivers are the ComQuest General Purpose Modem, the Raytheon Stand Alone Beacon Receiver, and the AN/ASC-30 Small EHF/SHF Terminal. The display system uses an RS-232 communications port to interface with the GPM and Raytheon receivers. The ASC-30 interface is through a parallel data acquisition card. In addition, the display system provides control of the GPM and Raytheon systems through RS-232 ports. The GPM uses COM2 of the PC, while the Raytheon system uses COM1 for both the control and beacon data ports.

For all hardware interfaces, the beacon display system has unique software interfaces to parse out the beacon data. The parsing software then calls the decode and display software to display the beacon configuration information on the PC screen. Figure 3 shows a DSCS III A display while Figure 4 shows a DSCS III B display.

The executable form of the program is named "SCT.EXE". To start the program, the user types **SCT** at the MS-DOS prompt. The user must ensure the program is on the current drive, or that a path name is used in front of the program name to tell DOS where the program is located. The program initializes the software variables and then displays the options menu shown

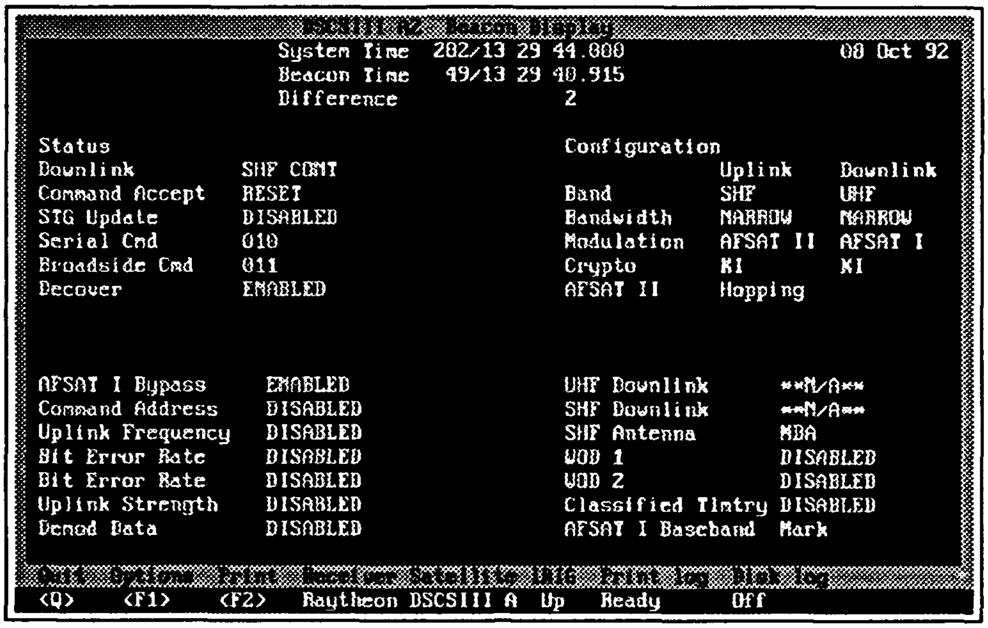

Figure 3. DSCS III A Beacon Display

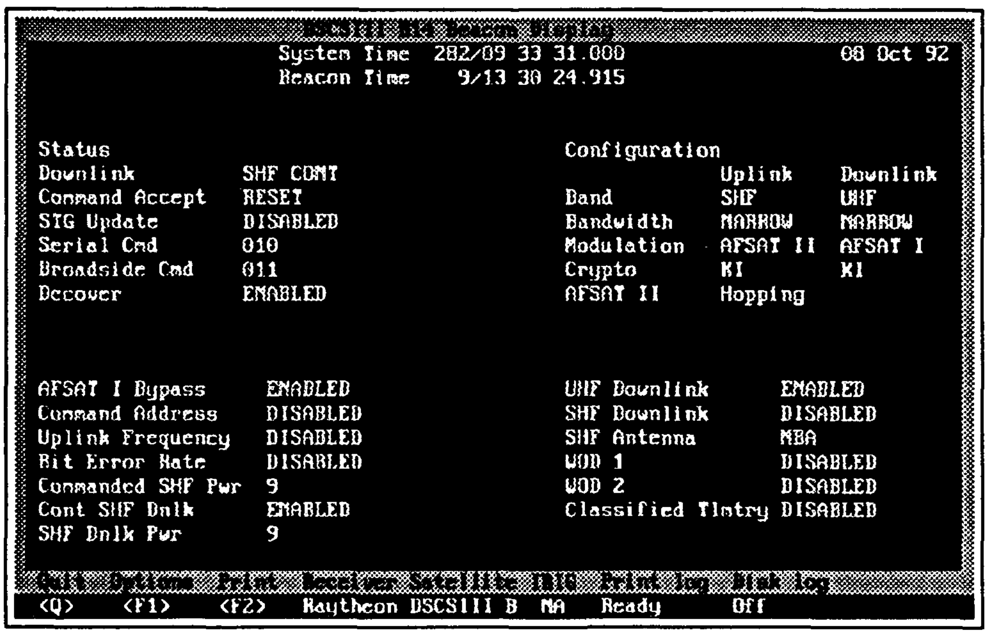

Figure 4. DSCS III B Beacon Display

in Figure 5. Note that the default receiver is the GPM. The pull-down menu options are discussed later in the chapter. To exit the options menu, the user presses the ESC key. The display system then executes all enabled options and modes. To change any option, or mode, the user presses the $< F1 >$ function key to bring up the options menu again. To exit the program, the user presses the $< Q >$ key to bring up the notice shown in Figure 6. When the user presses the $< Y >$ key, the program terminates by closing all files and ports and exiting to MS-DOS.

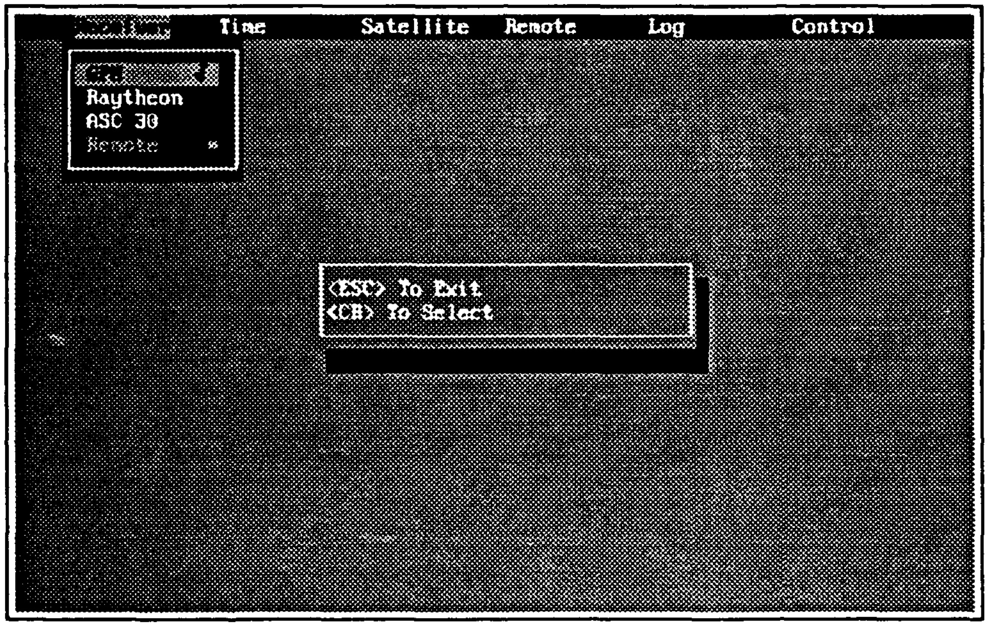

Figure 5. Options Menu

The primary function of the software is to decode and display the SCT beacon telemetry data delivered by one of the beacon receivers. The configuration display is shown in Figures 3 and 4. The information is displayed in five blocks. The first block is the decoded SCT clock time shown on line three. The second block is located in the upper left quadrant and is titled "Status". The information in this block was singled out specifically for SATCOM Group use when attempting to command the SCT. Block three is titled "Configuration" and is located in the upper right quadrant of the display. It contains pertinent information about the commanding configuration of the SCT. Blocks four and five are located in the lower left and lower right quadrants of the display screen,

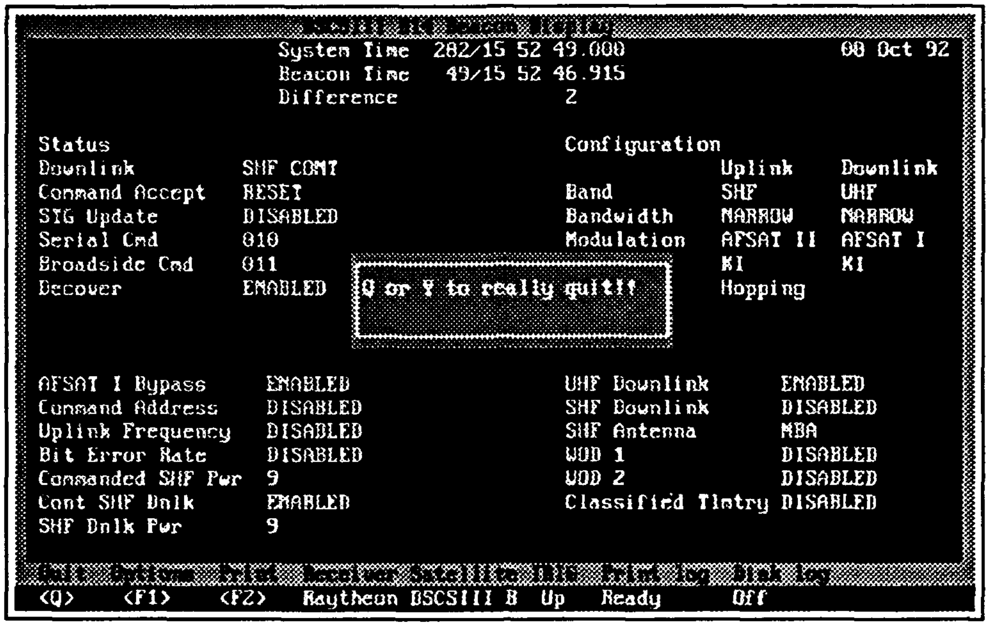

Figure 6. Beacon Display Quit Prompt

respectively. They display the remaining SCT beacon information. Block four is interesting because this is where the major differences between the DSCS III A and DSCS III B satellites are displayed.

In addition to the decoded beacon information, the SCT display system provides a satellite name, system time, system date, and system status information. As shown in Figure 4, the satellite name is on line one (DSCS III B14), while the system time and system date are located on line two. The display system status is located at the bottom of the display on lines 24 and 25. The status line shows the user commands available (Quit, Options, and Print) and the display systems configuration. The system configuration information consists of the beacon receiver used, the satellite model (DSCS III A or B), if the IRIG card is setting system time, and the system logging status. In addition, when the PC is equipped with a Bancomm IRIG card and the IRIG mode is enabled, line four of the PC display screen is used to display the difference, in seconds, between the system and the most recent decoded beacon time. Figure 3 shows the difference display when the system is using the GPM beacon receiver. The IRIG option, along with the other mode options, is discussed next.

3.3 System Modes

The software has additional user selected modes, or capabilities, available. The default, or "normal" mode, is when the display system is interfaced to one of the beacon receivers displaying the SCT information with the MS-DOS timer setting the system time. There are four additional modes: IRIG, Remote, Logging, and Control. These modes are enabled using the menu software activated by pressing the $< F1 >$ function key. Most modes are independent of each other. In other words, they can all be enabled at the same time. The exception occurs when the system is in the remote mode as the slave. In this case, the remote slave cannot implement the IRIG or Control modes because the remote master site is the system interfaced to the beacon receiver. The remote slave is interfaced to the telephone modem. The menu description section later in this chapter discusses the remote slave and remote master software in more detail.

As mentioned before, the IRIG mode enables additional software to use a Bancomm 630AT IRIG card to generate the system time, which is synchronized to a local IRIG B time standard [13]. The software generates an on-time interrupt, calculates the difference from the most recently decoded beacon SCT clock time, then displays the difference as shown in Figure 3.

The Remote mode consists of two additional pieces of software. The display system can be configured as either the remote master or the remote slave. The remote master is the system interfaced to one of the beacon receivers. The master software initiates the phone call to contact the slave at the remote site, sends the receiver type and satellite name to the remote slave, and then relays the beacon information as it comes in to the PC. The remote master continues to perform its own decoding and display duties while sending data to the slave.

When the remote slave software is started, it disables functions which are no longer appropriate (such as receiver selections) and waits for the remote master to call. When contact and synchronization is established, the remote slave obtains the receiver type and satellite name, and then operates in the normal mode.

The logging modes allow the user to capture the beacon display screen and save it to the printer or disk. The capture can be initiated by the user, a watchdog timer, or by the command accepted bit embedded in the decoded beacon telemetry data. All functions are independent and can be enabled at the same time.

The control mode will allow the users to control, or interact, with the GPM and Raytheon beacon receivers. Both receivers were designed to interface with a remote host, and the control mode software takes advantage of this. The control mode also provides start-up control and commands for both receivers.

More details the modes and functions are discussed next as each menu selection is detailed. The logical place to begin the description is with the main menu selections.

3.4 Menu Description

3.4.1 Main Menu Figure 5 shows the main menu selections of the SCT beacon display system. They are *Receiver, Time, Satellite, Remote, Log, and Control.* The left/right arrow keys move the highlighted main menu selection. At each menu item, the menu display software pulls down vertical sub-menu selections. The up/down arrow keys will move the highlighted vertical menu selection. A selection with a check mark indicates that particular menu item was selected and executed previously. To change the display parameters, the user moves the highlight to the desired pull-down menu item, then presses the carriage return $< CR >$ key. To exit the menu driver software, the user presses the $< ESC >$ key.

Menu items with an asterisk are disabled selections which can only be accessed when the SCT display system is in a particular mode, such as the remote slave mode. The following sections describe the pull-down menu items in detail, including how to get into the remote slave mode.

3.4.2 Receiver Menu The *Receiver* pull-down menu selections consist of four items; *GPM, Raytheon, ASC-30,* and *Remote.* The menu is also shown in Figure 5. Note the *Remote* selection

is disabled since it is in reverse video and followed by an asterisk. These selections allow the user to tell the system which SCT beacon receiver is connected to the PC. The three beacon receivers are the ComQuest General Purpose Modem (*GPM*), the Raytheon Stand Alone Beacon Receiver (*Raytheon*), and the AN/ASC-30 Small EHF/SHF Terminal (*ASC-30*). The system default receiver is the GPM. The pull-down menu item with the check mark tells the decode and display software which interface software to use and which port the data will be coming in from. Chapter II discusses the actual implementation details of the software and hardware.

The *Remote* selection is enabled only when the display system is in the remote slave mode. When in this mode, the remote master location controls the beacon receiver selection, so all other selections are disabled. Execution of the *Remote* menu item results in an error message.

3.4.3 Time Menu The *Time* menu item is used by the system to determine which time source will be used to set the system time. The user has two choices, *MS-DOS* and *IRIG* (Figure 7). The default is *MS-DOS*. If the user selects the *MS-DOS* menu item, the software disables the IRIG interrupt (if it was enabled), and then uses the MS-DOS timer interrupt to generate a one pulse-per-second (PPS) update of the system time display.

If a Bancomm IRIG card is installed in the PC, the user can enable the IRIG mode by selecting the *IRIG* item on the *Time* vertical menu. As the software initializes the IRIG card, Figure 8 appears. The card is used to generate a one PPS interrupt to display system time. In addition, the card time is used to calculate the difference between the IRIG time and the most recently decoded beacon clock time. The resolution of the difference calculation is in seconds.

3.4.4 Satellite Menu The *Satellite* menu item allows the user to select which DSCS III satellite the beacon receiver is demodulating. The pull down menu is shown in Figure 9. The menu selections list the possible DSCS III satellites available [19]. The software uses the information

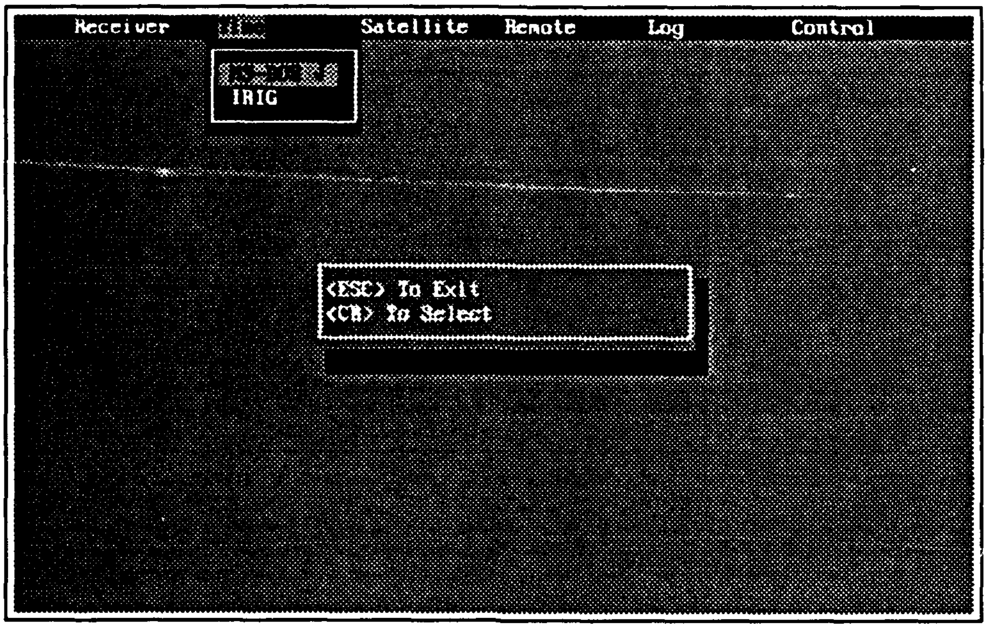

Figure 7. Time Menu

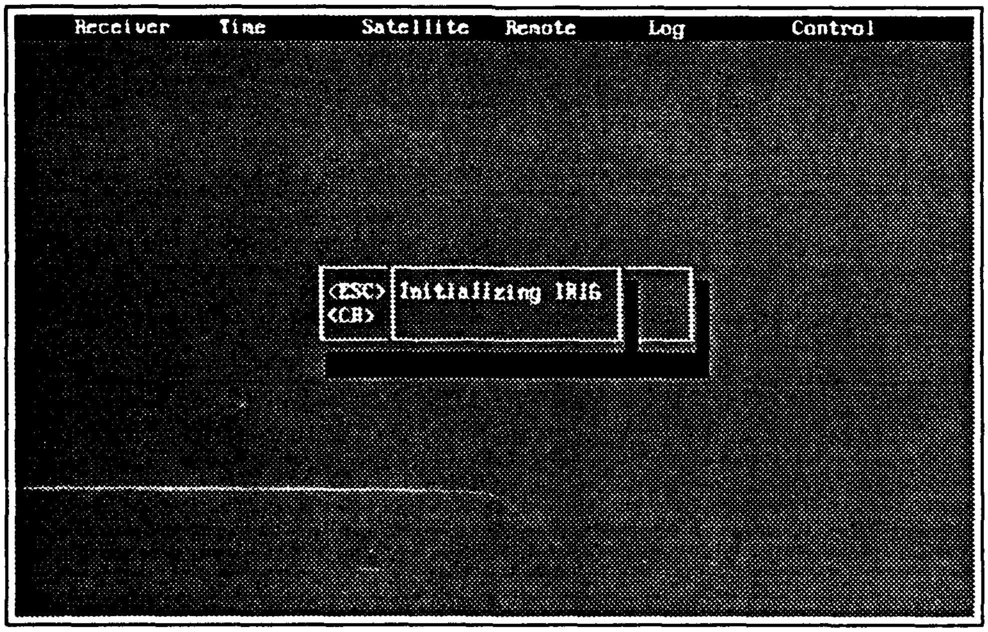

Figure 8. IRIG Initialization Notice

from this menu selection to determine which satellite name to display on the top line of the beacon information display window, as well as the type of satellite (DSCS III A *or* DSCS III B).

The *Other* selection allows the user to input a satellite name other than the ones listed on the menu. The entered name will be displayed on the top line of the beacon information window, but the satellite type is assumed to be DSCS III B. Figure 10 shows the data entry template for the user to enter the satellite name.

The disabled *Remote* item under the *Satellite* main menu option (Figure 9) is similar to the *Remote* selection shown in the *Receiver* pull-down menu. This selection is enabled only when the software is in the remote slave mode and the remote master is setting the satellite name. Selecting the *Remote* item when it is enabled results in an error message.

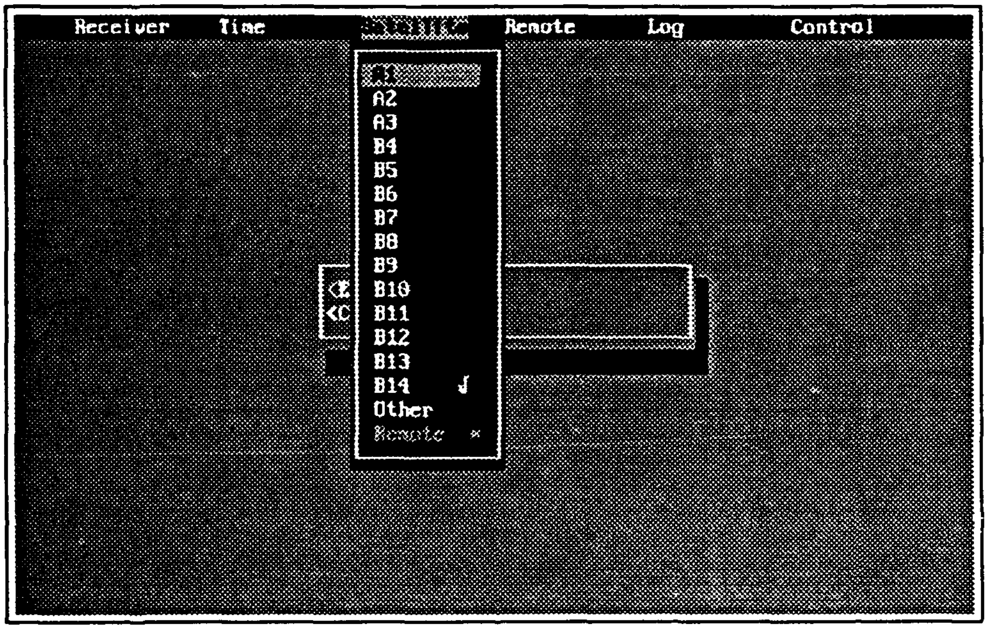

Figure 9. Satellite Menu

3.4.5 Remote Menu The *Remote* pull-down menu selections are shown in Figure 11. These software options allow for remoting the beacon display to a different location over a telephone link. The initial menu has four selections, two enabled and two disabled. The *Start Remote Master* item tells the display system it is now in the remote master mode and is expected to make contact with

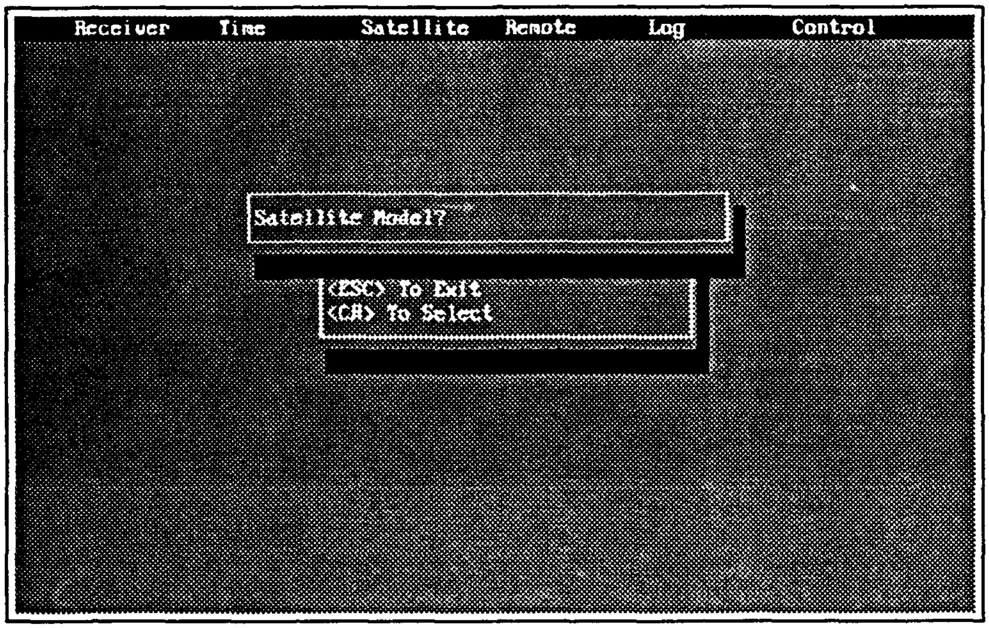

Figure 10. User-Selected Satellite Name

the remote slave and begin relaying data. The remote master software initializes the modem, then prompts the user for a phone number to dial as shown in Figure 12. Once the software has the phone number, it instructs the modem to dial the number and waits for the slave to answer. If the slave answers within 15 seconds, the slave and master exchange synchronization messages to establish contact. The remote master code displays the message shown in Figure 13 while waiting for the slave software to answer. Once synchronization has been obtained, the master software sets a global flag telling the port software to also send beacon data to the remote site. If synchronization fails, an error message is displayed and the master software returns to the main menu.

If the slave fails to answer the phone within 15 seconds, the timeout message shown in Figure 14 is displayed. The user can continue to wait, or exit the master software.

Once the system is in the remote master mode, the only menu selections enabled in the *Remote* pull-down menu is the *Stop Remote Master* item. If the user selects this menu item, the software instructs the modem to hang up the phone and reset. Once the modem is reset, control is returned to the receiver software in the normal beacon display mode.

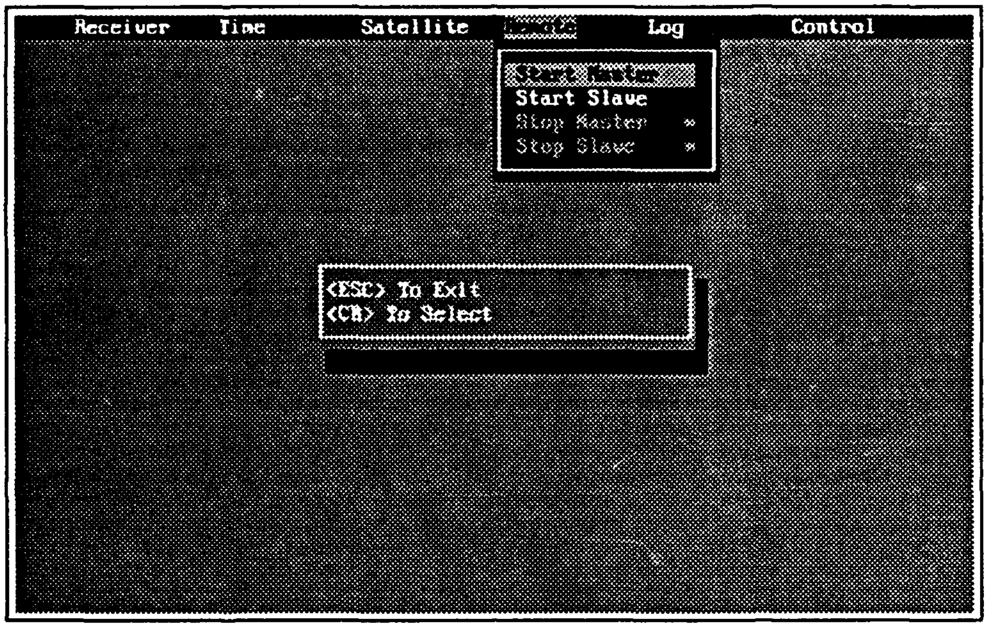

Figure 11. Remote Menu

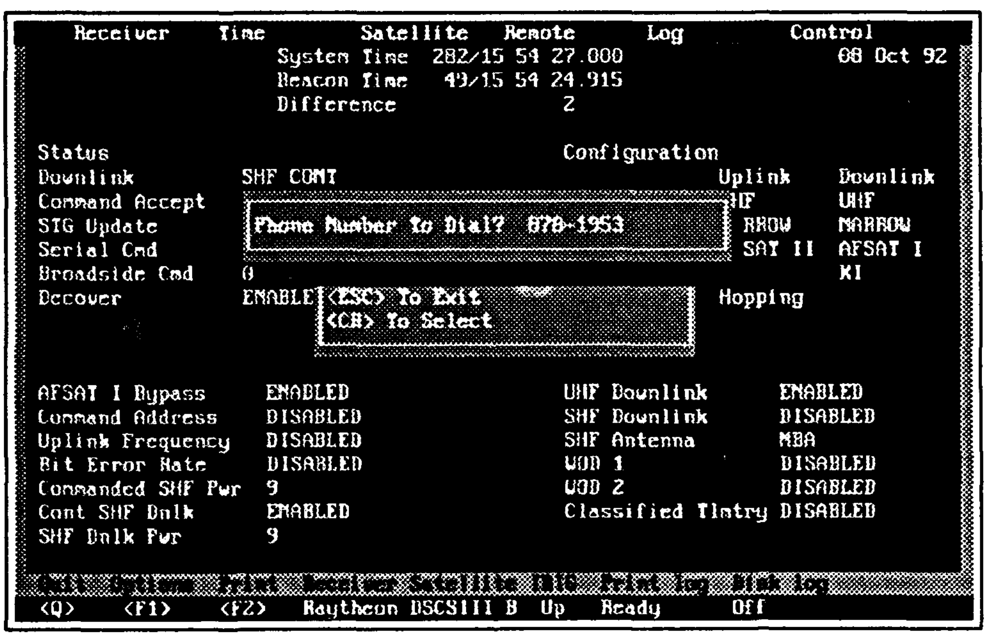

Figure 12. Phone Number Prompt

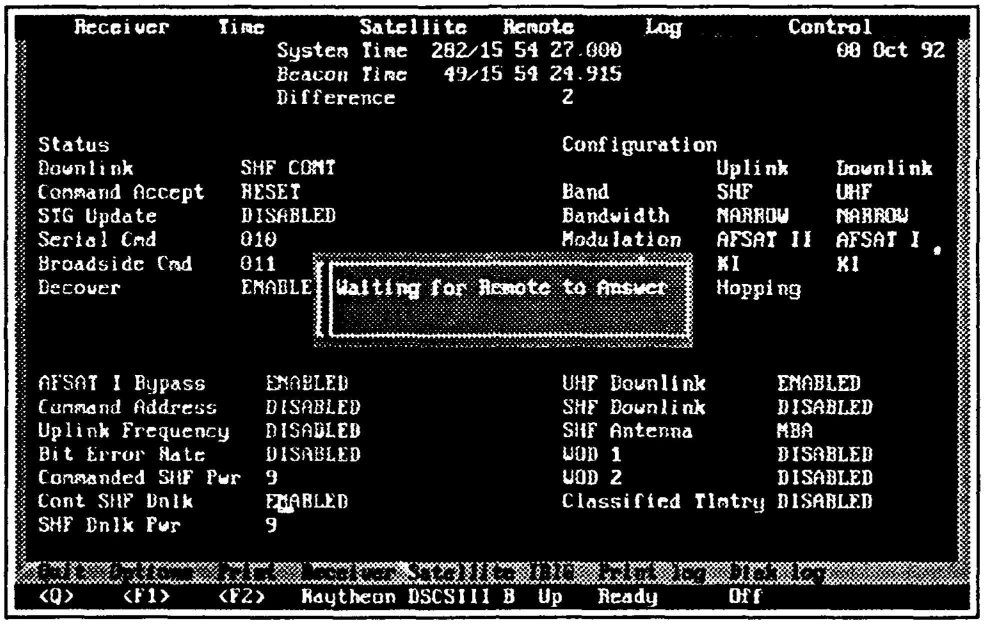

Figure 13. Remote Master Wait Notice

The software operating at the remote slave site must be in the remote slave mode. There are two ways to get the display system into this mode. One is for the user to select the *Start Remote Slave* item in the *Remote* pull-down menu. The other is to specify remote slave at system start-up by typing **SCT REMOTE** at the MS-DOS prompt. The remote slave software initializes the modem and puts it into the auto-answer mode. It then waits for the master to initiate a phone call. When the master does call, the slave software initializes the synchronization sequence after a carrier is detected. Once the proper handshaking sequence is accomplished, the master sends the beacon receiver type (GPM or Raytheon) and the satellite name to the slave. The remote master software cannot be used if the ASC-30 is the beacon receiver due to system configuration problems discussed in Chapter II. The slave then starts the proper parsing software and program control is turned over to the receiver software.

To exit the remote slave software, the user selects the enabled *Stop Remote Slave* menu item in the *Remote* pull-down menu. The slave software instructs the modem to hang up the phone

and reset. When reset is completed, the slave software returns control to the decode and display software where the user can exit the program by typing the $< Q >$ then $< Y >$ keys.

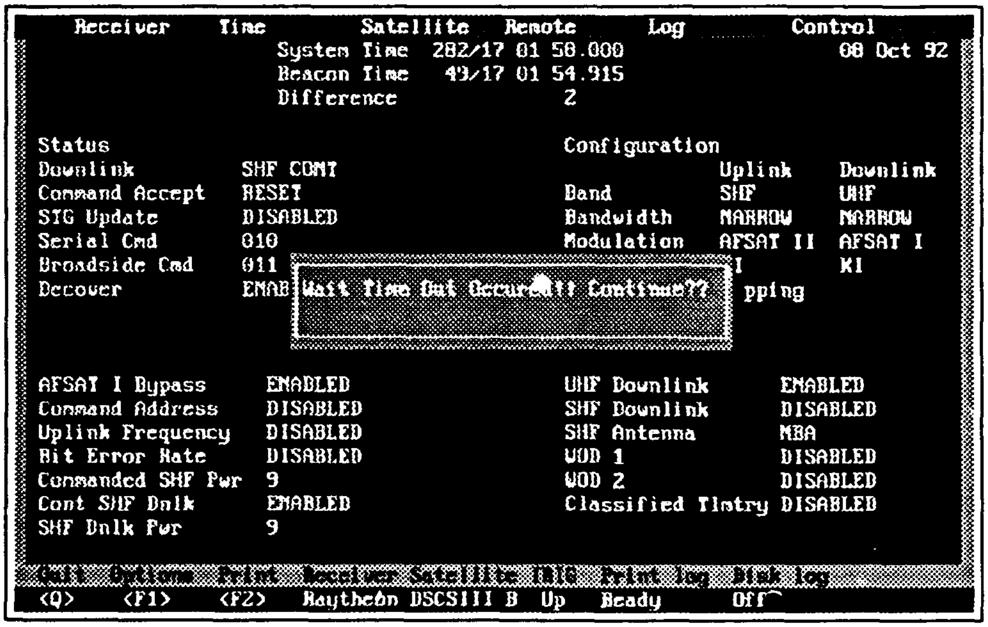

Figure 14. Remote Master Contact Timeout

3.4.6 Log Menu The next main menu selection is the *Log* item. The pull down menu is shown in Figure 15. As can be seen, there are four selections: *Trigger Print, Timed Print, Trigger Disk,* and *Timed Disk.* The selections put the display in different logging modes, depending on which menu items are selected. The menu selections with the check marks are the logging modes enabled.

The *Timed Print* and *Timed Log* items allow the user to select timed intervals to print, and/or save to disk, the beacon information display. Figure 16 shows the user prompt screen for the interval timer input. The user has a choice of 1 to 120 minute intervals. When the interval timers are enabled, the system captures, then prints (or saves) an initial beacon screen three seconds after the item is selected. The delay allows the decoding software to capture a current beacon configuration and display it to the screen.

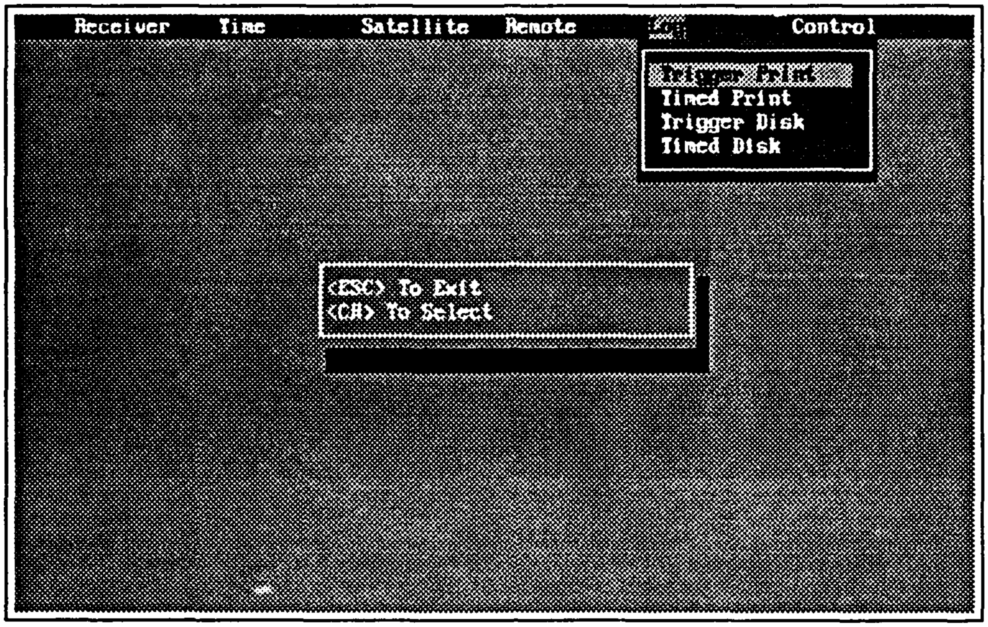

Figure 15. Log Menu

The *Trigger Print* and *Trigger Disk* logging modes tell the decode software to monitor the command accept bit in the beacon information (see Chapter II). When the bit toggles to TRUE, the decode software signals the system to print (or save to disk) the current beacon display.

To disable any active logging modes, the user selects the desired menu item and presses $< CR >$. The selection will toggle and disable the logging mode.

3.4.7 Control Menu The pull-down menu selections under the *Control* main menu item allow the user to interface with the GPM and Raytheon beacon receivers. The selections are shown in Figure 17. The *GPM START* menu item opens the window shown in Figure 18. This software package sends the proper configuration commands to the ComQuest GPM for DSCS III beacon operation. The commands are described in the GPM reference manual [14]. Once the commands are sent, the software waits for three seconds to allow the user time to check the GPM responses for errors. If an error has occurred, the *GPM Window* menu item can be used to interact with the GPM through its remote control port. Figure 19 shows the GPM control window. The *GPM*

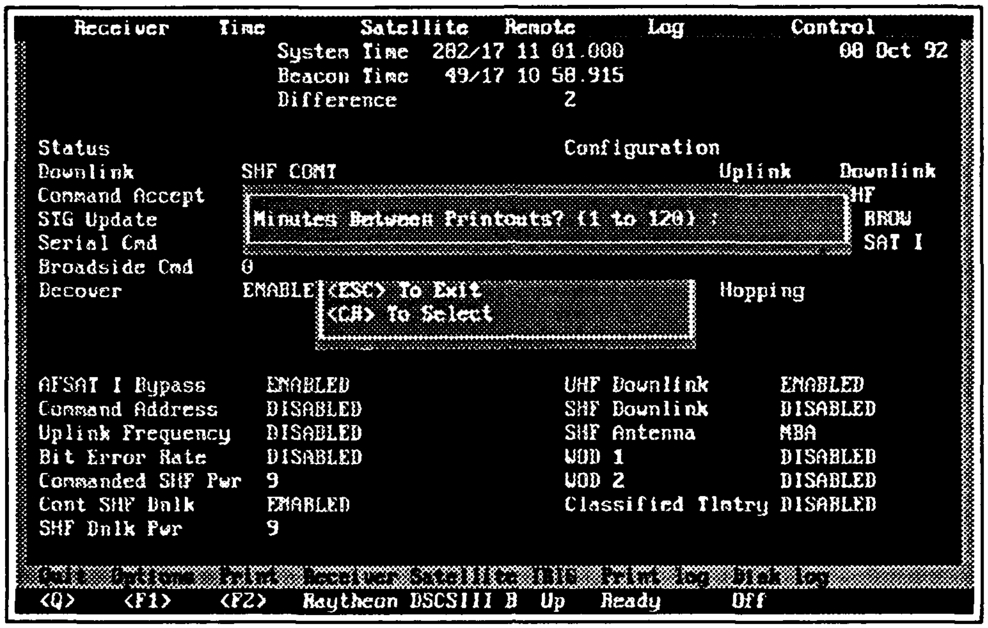

Figure 16. Logging Timed Print User Prompt

Window software sends the user keystrokes to the GPM and prints the GPM responses to the PC screen in the window. It was installed to initialize and check the GPM for proper operation.

The *Ray Start* menu item provides the software required to bring up the Raytheon Stand Alone Beacon Receiver. Figure 20 shows the Raytheon start window. To use the window, the user selects the menu option, then restarts the Raytheon beacon receiver. When the beacon receiver has completed its self test, the software prompts the user to select a satellite (Figure 21). The proper commands are then sent to the beacon receiver and program control returns to the menu software.

The last *Control* menu item is *Ray Offset*. It allows the user to adjust the Raytheon beacon receiver frequency offset. This may be required if a particular satellite has high Doppler shift. The frequency shift window is shown in Figure 22. The user enters the desired frequency shift (in Hertz) and the software generates the required commands to offset the Raytheon beacon receiver center frequency.

3.5 Summary

This chapter presented an operational description of the beacon display software. The function of each menu item was described in terms of how the selections affect the operation of the beacon display software and in which modes they put the system. The next chapter presents the start-up procedures required to bring up the beacon receivers and the SCT beacon display software.

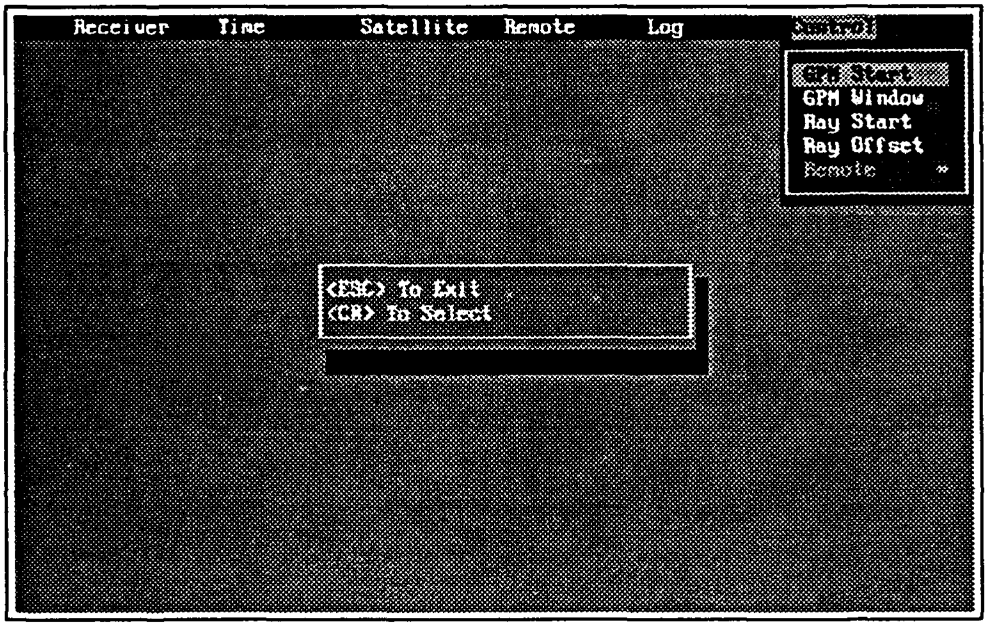

Figure 17. Control Menu

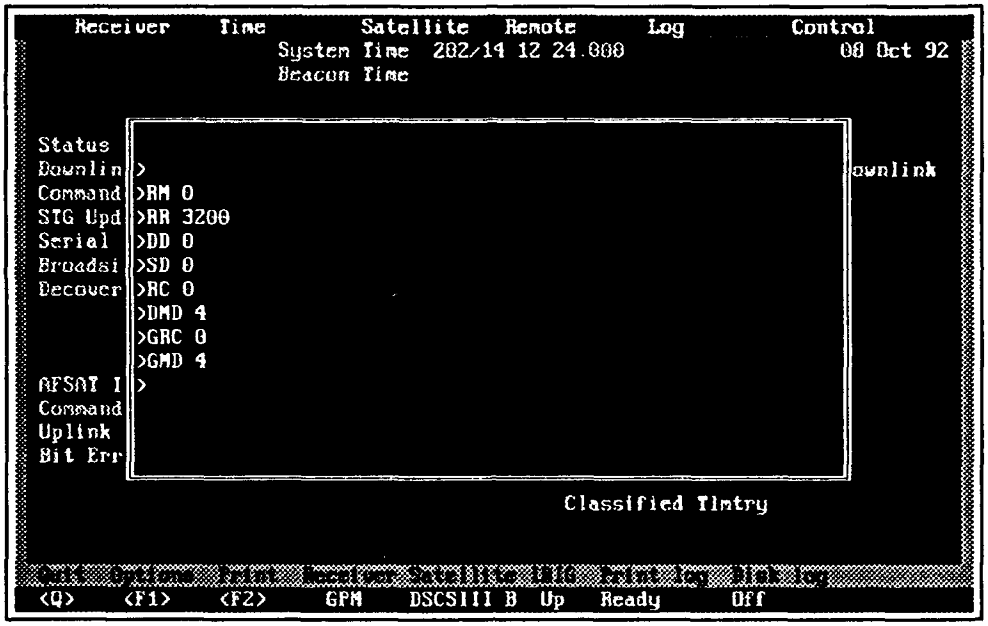

Figure 18. GPM Start Window

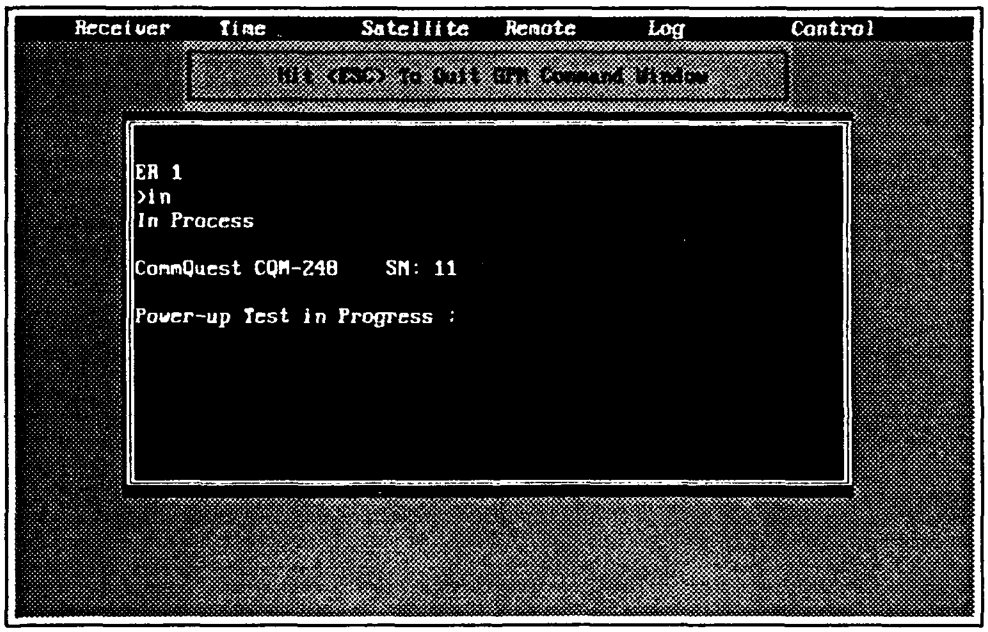

Figure 19. GPM Control Window

Figure 20. Raytheon Start Window

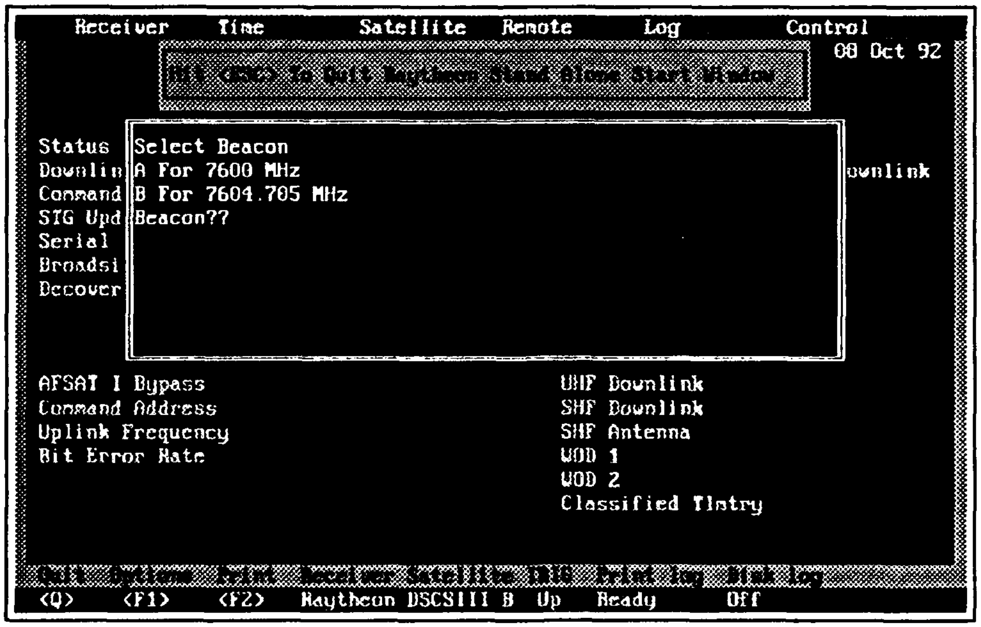

Figure 21. Raytheon Satellite Type Prompt

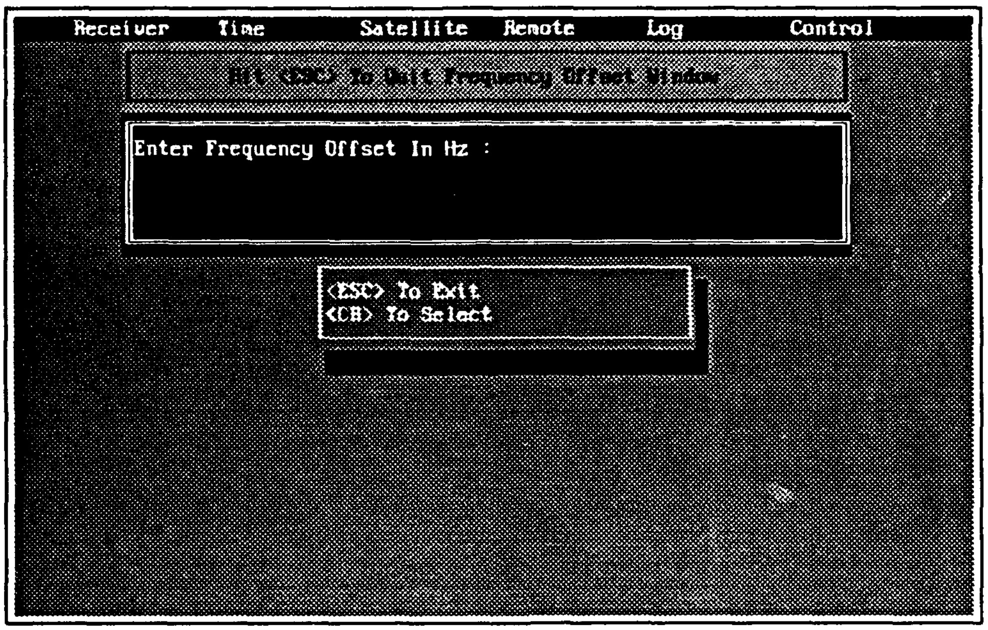

Figure 22. Raytheon Offset Window

IV. Start-Up Procedures

4.1 Introduction

This chapter details the start-up procedures of the SCT beacon display system. The guiding purpose of the chapter was to provide all the necessary information so users can start up the required equipment and display the SCT configuration. There are three sections to the chapter. The first describes how to bring up the beacon receivers, the second discusses how to configure the beacon display software for desired operation, and the third provides troubleshooting instructions should there be any problems.

4.2 Beacon Receiver Start-Up Procedures

There are three beacon receivers which may be used by the beacon display system. They are the ComQuest General Purpose Modem (GPM), the Raytheon Stand Alone Beacon Receiver (SABR), and the AN/ASC-30 Small EHF/SHF Terminal. Some familiarity with the ASC-30 system is assumed in the start-up procedures. Specific details on how to interconnect and configure the ASC-30 is provided in [1].

The following section provides itemized procedure lists on how to bring up the beacon receivers for stand alone use, or for use as a RF front end for the GPM.

4.2.1 AN/ASC-30 Start-Up This section describes how to bring up the ASC-30 for use in the stand alone mode of operation. The section on the GPM start-up describes how to use the ASC-30 as the front end for the GPM.

The minimum configuration of the system consists of the Frequency Terminal and Control Unit (FTCU), a SHF receiver, a Remote Control Unit (RCU) or RCU emulator, and a 5 MHz reference. The following list describes how to bring up the ASC-30.

1. Interconnect and power up the ASC-30 equipment.

2. Load the FTCU software.

3. To save time, locate the SHF antenna/LNA equipment where the desired satellite is visible while the FTCU is loading.

4. Connect the SHF antenna to the LNA input.

5. Connect the LNA output to the SHF input on the ASC-30 receiver.

6. Connect one SHF receiver 70 MHz high level output to the FTCU receive IF input.

7. Power up the LNA.

8. Once the FTCU software is loaded, load the beacon patch (if used as a stand alone), and the upgrade receiver patch (if using an upgrade receiver).

9. Start the FTCU software (send G1000).

10. Using the RCU (or emulator), configure the ASC-30 as listed below.

    (a) Acquisition $\frac{P_r}{N_0} = 40$.

    (b) Noise measurement = ON.

    (c) Select Beacon Receiver (Receiver 1 or 3).

    (d) Receive Frequency = 7604.705.

    (e) Receive Offset (RCDO) = 882.

    (f) Execute the configuration.

    (g) Perform an acquisition to set the FTCU Automatic Gain Control (AGC).

11. Configure the spectrum analyzer as shown in Table 15.

12. Connect the other SHF receiver 70 MHz high level output to the input of the spectrum analyzer.

13. Point the SHF antenna to get a spectrum analyzer display similar to the one shown in Figure 23 with the signal peak 8 to 10 dB above the noise.

14. Perform another beacon acquisition and the RCU display should indicate the FTCU has acquired and phase-locked to the beacon signal.

15. The 70 MHz connected to the spectrum analyzer may now be used as the input to the GPM (connector J5).

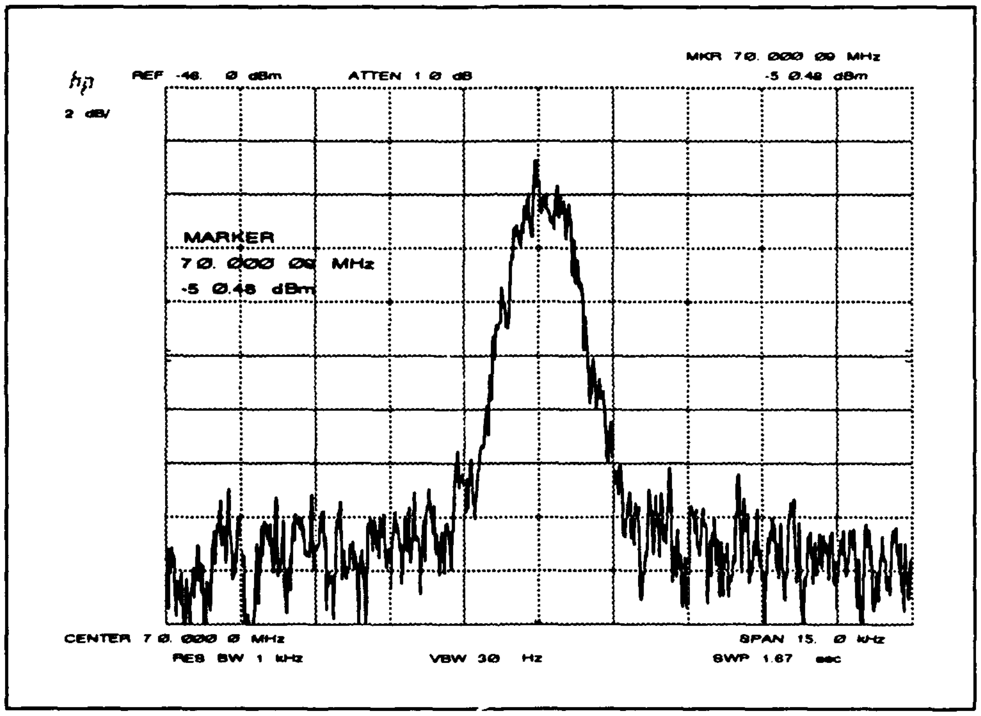

Figure 23. ASC-30 Spectrum Analyzer Display

Table 15. Spectrum Analyzer Settings

Adjustment	Value
Center Frequency	70 MHz
Frequency Span	15 KHz
Resolution Bandwidth	1 KHz
Video Bandwidth	30 Hz
Scale	2 DB/DIV
Reference Level	-46 DBM

4.2.2 Raytheon SABR Start-Up This section presents the start-up procedures for the Raytheon Stand Alone Beacon Receiver. This system may also be used in a stand alone mode, or as a RF front end for the GPM.

There are two PC programs available to configure the Raytheon system. The first is the Pascal software delivered with the system by Raytheon. The second is software embedded in the beacon display system. If the Raytheon system is going to be used as an RF front end for the GPM, then either program may be used. If, however, the system will be used as a stand alone to feed data to the display software, the embedded start software *Ray Start* in the display system must be used. This is required because the receiver control information is sent on the same RS-232 link as the demodulated beacon data. If the Raytheon Pascal software is used to bring up the system, a cable change to another machine running the beacon display system causes the receiver to lock up.

Following are the instructions which detail how to bring up the Raytheon beacon receiver.

1. Connect the PC COM1 port to the beacon receiver RS-232 port (Connector J3 in rear).
2. Power up both the PC and the Raytheon beacon receiver.
3. Press the reset switch on the receiver to allow the 10 MHz reference oscillator to warm up and stabilize.
4. Locate the SHF antenna/LNA assembly where the desired satellite is visible.
5. Cable the SHF antenna to the LNA input.
6. Cable the LNA output to the receiver SHF input.
7. Power up the LNA.
8. Bring up the control software on the PC. For the Raytheon Pascal program, type "SCT_MON". For the embedded control software, type "SCT" at the MS-DOS prompt. Select the *Control* option and enable the *RAY START* menu item.

9. Press the restart switch on the beacon receiver again. The control software should be displaying the receiver self-test procedures.

10. Select satellite "B" when self-test is complete.

11. Configure the spectrum analyzer as shown in Table 15.

12. Connect the 70IF Test2 output (BNC connector in front) to the spectrum analyzer input.

13. Adjust the analyzer reference level to center the display.

14. Point the SHF antenna to get a spectrum analyzer display similar to the one shown in Figure 24.

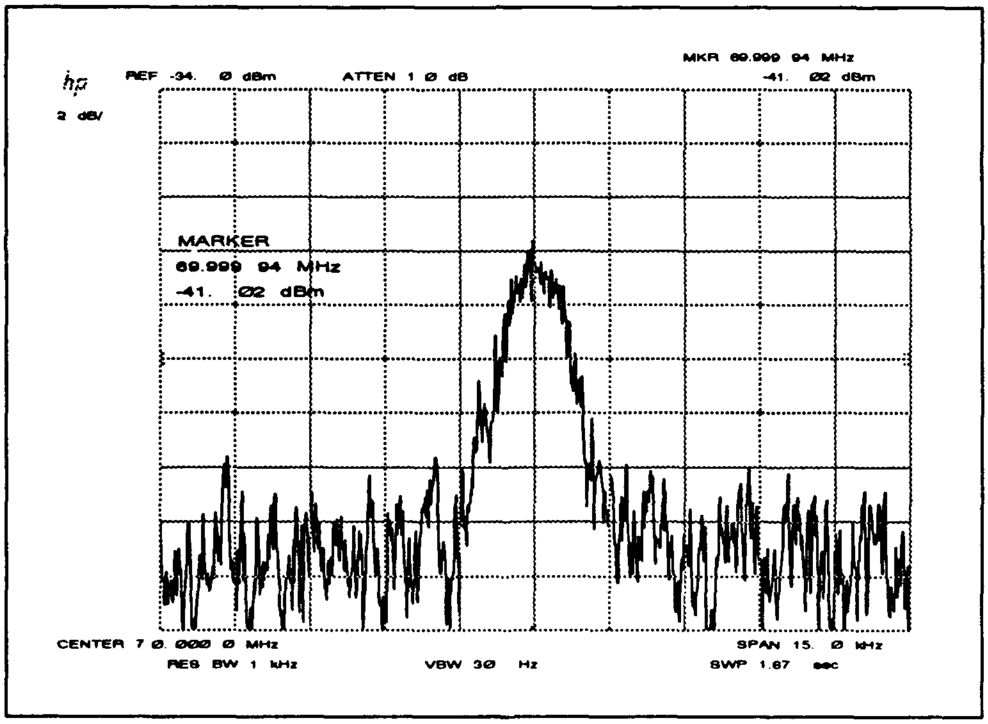

Figure 24. Raytheon Spectrum Analyzer Display

Once the procedures are completed, the demodulator (GPM or Raytheon) should acquire and phase lock to the signal within 10 minutes. If the receiver does not acquire, reset the Raytheon system to repeat the self-test and start up procedures. This may be required if the 10 MHz reference oscillator has not stabilized.

4.2.3 GPM The GPM is a demodulator only and depends on external equipment to provide the beacon signal on a 70 MHz intermediate frequency (IF). Both the ASC-30 and the Raytheon stand alone can provide the input. For the ASC-30, the required 70 MHz is available at the SHF 70 MHz high output (pins E or F of RF connector). The Raytheon 70IF Test2 BNC connector on the front panel will also provide the required signal level. The desired source must be cabled to the GPM RXIF connector (J5). The 70 MHz signal must be similar to the ones shown in Figures 23 and 24. The following instructions list the additional steps required to configure the GPM for beacon operation.

1. Bring up the selected RF front end system.
2. Connect the 70 MHz receiver output to the GPM RXIF input, connector J5, at the rear of the receiver.
3. Connect the PC COM1 port to the GPM beacon data port, connector J6A with attached 2-to-9 pin adapter cable.
4. Connect the PC COM2 port to the GPM remote host RS-232 port, connector J1.
5. Power up the PC.
6. Type "SCT" at the MS-DOS prompt to bring up the beacon display system.
7. Select the *Control* option and enable the *GPM Window* menu item.
8. Power up the GPM. The GPM power up self-test messages should now be appearing on the PC screen.
9. Once the GPM has passed self test, exit the GPM window screen and select the *GPM Start* menu item under the *Control* option. The software will now send the proper commands to configure the GPM for beacon operation.
10. Press the < *ESC* > key to exit the options menu.

The GPM should now acquire (indicated by the RXSYNC light on the front panel) and send data to the PC for display. Once the display system is receiving data, the software can be reconfigured for desired operation.

4.3 Beacon Display System Control

This section describes how to get the display system software into the desired configuration.

4.3.1 Receiver The user must tell the software which receiver is going to provide the beacon data to be displayed. Each receiver has specific hardware and software interfaces to the PC.

4.3.1.1 GPM The display system defaults to GPM as the beacon receiver. COM1 must be connected to the GPM beacon data port. COM2 must be connected to the GPM remote host port if the *Control* menu items will be used. The GPM must also be interfaced to a 70 MHz IF provided by the ASC-30 or Raytheon systems. Once all connections have been made, select the *GPM* entry from the *Receiver* options menu.

4.3.1.2 Raytheon To use the Raytheon beacon receiver as the source of beacon data, the PC COM1 port must be connected to the control port of the beacon receiver. The Raytheon system must have acquired the beacon signal before it will send data to the PC. Acquisition is indicated by the ACQ light on the receiver front panel. Once the ACQ light is on, select the *Raytheon* item under the *Receiver* option and the display system will display the data sent by the Raytheon beacon receiver.

4.3.1.3 ASC-30 To use the ASC-30, the beacon data and clock signals from the beacon receiver must be level-shifted from RS-232 levels to TTL levels. An MC 1489 line receiver installed in the SABR DAC interface provides the level shifting. The converted signals must be connected to a DIO-24 parallel input card developed by Industrial Computer Source [12]. Table 16 shows the

pins used on the card and the SABR DAC interface 9-pin male connector. The PIO card shares the COM2 interrupt, therefore, the ASC-30 interface cannot be used as the beacon receiver for remoting purposes.

Once all interface requirements are met, the ASC-30 can be powered up and configured for beacon use. The FTCU must acquire and phase lock to the signal before proper data is sent to the PC interface. The user selects the *ASC-30* menu item to tell the display system the ASC-30 will be sending the data to the PC.

Table 16. DIO-24 Pins Used

DIO-24 Pin	SABR DAC 9-Pin (Male)	Signal Name
37	1	800 BPS Data
1	2	Beacon Clock
19	7	Ground
1 wired to 20		INT Enable Jumper

4.3.2 IRIG The display has the capability of presenting the difference between the decoded beacon clock and a local IRIG B time standard. To enable the IRIG mode, the PC must have a Bancomm 630AT time card installed [13]. The card must be cabled to a local IRIG B time reference. Recommended is a GPS satellite clock, or the DATUM Model 9815 rubidium portable clock. Table 7 in Chapter III shows the required configuration of the card for proper interrupt operation. To use the installed card, select the *IRIG* menu item under the *Time* option. The display system will begin to display the difference between the IRIG B 1 PPS interrupt and the most recent decoded beacon clock time. The resolution is in seconds.

NOTE!!

This time difference is dependent on the beacon receiver used. When the GPM is used, the displayed difference is 5 seconds if the SCT is "on time". "On time" is defined to be when the Command Post Modem/Processor can command the SCT when set to IRIG system time. For the Raytheon and ASC-30 receivers, the displayed difference is 2 seconds.

4.3.3 Remoting

4.3.3.1 Remote Location To successfully remote the SCT beacon display, the remote site must have a copy of the SCT.EXE program, a PC to run it on, a phone, and a Hayes-compatible modem. Recommended is the ZOOM model FX9624 modem. The modem must be connected to COM1 of the PC and the phone line.

To bring up the remote slave software, the user types **SCT REMOTE** at the MS-DOS prompt, or selects the *Start Remote Slave* selection under the *Remote* option if the display system is already running. The software initializes the modem and instructs it to auto-answer the phone. The auto-answer mode will be indicated by the lighted AA indication on the front of the ZOOM modem. The software then waits for the master to call. When the phone rings, the software will answer the phone, provide synchronization information to the master software running at the beacon receiver location, and then display the beacon data to the PC screen as it is relayed by the master software. Both the remote and beacon receiver locations display the SCT configuration once contact and synchronization has occurred.

To stop the remote slave software, select the *Stop Remote Slave* item under the *Remote* options menu.

4.3.3.2 Beacon Receiver Location To remote the beacon display running at the beacon receiver location, the display system must be connected to a beacon receiver and displaying the SCT configuration. COM2 of the PC must be connected to a Hayes-compatible modem. Recommended is a ZOOM model FX9624 or CTS DATACOM model 2424ADH modem. The remote slave software must be waiting for the master software to call for proper synchronization to occur. To initiate the remote master call, the user presses $< F1 >$ to pop-up the options menu, and then selects the *Start Remote Master* item under the *Remote* option. The software initializes the modem, prompts the user for a phone number, and then has the modem dial the number. Once the remote slave software

has answered the phone and a carrier is detected, the master and slave software synchronize and return to the decode and display software. The master software now relays the beacon data to the slave site as it continues to display the beacon configuration.

To stop the master software, the user again presses $< F1 >$ to bring up the options menu. The *Stop Remote Master* under the *Remote* menu item tells the software to hang up the modem and return to the decode and display software.

4.3.4 Logging The beacon display software is designed to copy the configuration shown on the PC screen to the attached printer or to the currently active disk drive. The names of the saved files correspond to the day, hour, and minute of the captured screen with a .cap extension.

Three triggers are available for both the printer and disk. The first is a user trigger accessed by pressing the $< F2 >$ key for an immediate print capture, or $< F4 >$ for disk capture. The second trigger is a watchdog timer enabled by selecting the timed print or timed disk items under the *Log* option of the main menu. The user can select from 1 to 120 minutes between prints or disk saves. To disable the timed captures, the user toggles the enabled selections by selecting them again. The final trigger is enabled by selecting the *Trigger Print* and/or *Trigger Disk* items under the *Log* menu. These triggers item will capture a beacon screen (if enabled) when the command accept bit of the decoded beacon display is toggled to TRUE.

The only requirements to operate the logging modes are that the printer be connected and on-line (with paper installed), and there is room on the logged disk for MS-DOS to write the captured data. Unpredictable results occur when MS-DOS cannot write the beacon data to the disk (usually a program crash).

4.4 *Troubleshooting*

Tables 17 and 18 list possible problems which may be encountered when bringing up the SCT beacon display system. The tables show a symptom, list a possible cause or causes, then recommends a solution to solve the problem.

4.5 *Summary*

This chapter summarizes the information required to bring up and configure the beacon receivers and beacon display system for desired operation. Included were discussions on the beacon receivers themselves, and how to enable the various options and modes provided by the SCT beacon display system. Also provided was a troubleshooting guide should start-up problems occur.

Table 17. Troubleshooting Guide

Symptom	Cause	Solution
GPM		
No RXSYNC indication.	Improper GPM Configuration.	Using the GPM window, initialize the GPM (IN command). Using the GPM Start option, put GPM into beacon mode.
	Insufficient IF signal.	Ensure the 70 MHz signal looks like Figure 23 or 24 and that it is present at the RXIF GPM input, connector J5 at rear of receiver.
No GPM Control Response.	Improper Cabling.	Ensure GPM remote host connector(J1) is cabled to PC COM2. Insert a null-modem to check for proper DCE/DTE connections. Use the GPM front panel controls to check handshaking settings. Proper settings are shown in Table 19.
Raytheon		
No Response in Ray Start Window.	Improper Cabling.	Ensure Raytheon PC connector (J3) is cabled to the PC COM1 port. Insert a null-modem to check for proper DCE/DTC connections.
No ACQ Indication.	No phase lock.	Ensure the 70 MHz test signal looks like Figure 24. Reset beacon receiver and perform self-test and startup again.
ASC-30		
No Phase Lock on RCU Screen.	Improper Setup.	Check ASC-30 cabling, then reload the FTCU and re-acquire
	Poor Reference.	Check 5MHz ASC-30 reference for proper levels.
	Insufficient IF signal.	Ensure the 70 MHz IF looks like Figure 23.
Display system cannot find sync bit.	Improper cabling	Check level shifter and connections. Use oscilloscope to ensure the TTL level data and clock are getting to the PIO card in the PC.
	PIO HW Settings.	Check PIO DIP switch settings to see if they match the requirements shown in Table 20.

Table 18. Troubleshooting Guide (Continued)

Symptom	Cause	Solution
IRIG		
Card initialization error.	IRIG HW Settings.	Check IRIG card DIP switch settings to see if they match the requirements shown in Table 20.
Card initializes, but no beacon clock offset display.	IRIG card.	Turn off the PC for at least one minute to reset the IRIG card.
Beacon Display		
Unwanted characters on screen.	Display Software.	Press < *F1* > then < *ESC* > to redraw the beacon screen.
Remote		
Cannot configure Hayes-compatible modem.	Cabling.	Must be Hayes compilable modems. Ensure COM1 is connected to the modem for the remote site, COM2 for the local site. Also check the modems for connections to the phone lines.
	No Site Synchronization.	Use the ZOOM modems at both locations, or use the CTS DATACOM at the master location with a ZOOM at the slave.

Table 19. GPM Interface Settings

Port	Baud	Data	Parity	Stop
Com1	9600	7 Bit	Odd	0 Bits
Com2	1200	7 Bit	Odd	0 Bits

Table 20. PC Interface Card Summary

Card	IRQ	I/0 Address	Interrupt	Vector
Com1	IRQ4	0x3F8-3FF	0x0C	0x30-33
Com2	IRQ3	0x2F8-2FF	0x0B	0x2C-2F
PIO	IRQ3	0x2F8-2FF	0x0B	0x2C-2F
LPT1	IRQ7	0x370-37F	0x0F	0x3C-3F
IRIG	IRQ5	0x320-32F	0x0D	0x34-37

V. Conclusion

5.1 Summary

This thesis provides a SCT beacon telemetry display system for the General Purpose Modem. The system is hosted on an IBM compatible personal computer. In addition to the basic beacon display, the following capabilities were implemented.

- System interfaced to the Raytheon Stand Alone Beacon Receiver.
- System interfaced to the AN/ASC-30 Small EHF/SHF Terminal.
- Capability to remote the display to another location.
- Capability to determine the SCT clock offset from an IRIG UTC source.
- SCT configuration display recording capability.
- User selected satellite name.

The required hardware was specified, procured, and installed. Software was developed in C using Borland's Turbo C++ Professional development system.

5.2 Conclusions

The primary goal of this thesis effort was to provide additional capability for the SATCOM Group of the Wright Laboratory when using the single channel transponder. It is believed the effort has been successful. The system provides improved SCT configuration documentation over current capability through the logging modes. It also provides a means of determining the SCT clock offset using the IRIG system timer without having to command the satellite. In addition, the system can be run on any properly configured IBM compatible personal computer. This removes the dependence on an aging, one-of-a-kind, SCT terminal for obtaining a SCT configuration. The

final enhancement is the ability to remote the SCT display to another location. Based on experience with the SATCOM Group, this may prove to be a most useful feature.

5.3 Recommendations

5.3.1 SCT Monitor System Significant additional software could be added to allow remote control of the entire system to include pointing the antenna, remote start-up of the beacon receivers, and remote upload of the beacon configuration. With the enhancements, the system could automatically come up and capture a beacon configuration unattended and on a regular basis. This would be useful for long-term tracking of SCT information such as SCT clock drift.

5.3.2 IRIG The current SCT beacon clock offset calculation resolution is only to the nearest second. The IRIG card can provide accuracy down to microseconds. A special beacon clock offset mode could be added which would do nothing but decode the beacon telemetry clock bits and determine a more accurate time offset from UTC. Measurements using the CPM/P as a reference terminal could quantify the processing delays of the beacon receiver and PC code. With these measurements, the resolution of the offset measurement to be reduced to milliseconds.

5.3.3 Remote The remoting software could be enhanced to allow the remote slave to call the master when it is running. This would allow a user to set up the beacon receiver and display system at one location, then go to the remote location and bring up the slave without an operator at the beacon receiver. In addition, support could be added to remote the display over a satellite link as well as the phone link. This would allow the beacon display to be remoted to the SATCOM airborne testbed while it is in flight.

5.3.4 Additional COM Ports The PORT.C software could be modified to allow more COM ports (COM3, COM4, etc). This would allow the remote and GPM control functions to operate at the same time and eliminate cable changes. Additional ports would allow the addition of mouse

driver software so the menu system could use the mouse as well as the keyboard to select user options, remote control of the ASC-30, and even control of an antenna pointing system.

5.3.5 Modem Software The remote slave software is not able to take the modem out of the auto-answer mode unless the master hangs up first. Research into the modem control strings may yield a solution to the problem.

5.3.6 File Software The file logging software could be enhanced to ensure the system does not crash when MS-DOS cannot write the captured screen to the disk.

5.3.7 MS-DOS Timer A set time function could be added to the MS-DOS timer software. The DOS timer tends to be significantly different from UTC time. The user can set the time using the DOS utility, but an integrated function in the display software would allow the user to set the time without exiting the program.

5.4 Lessons Learned

There are two lessons to be passed on. The first is that if a thesis topic appears to be a software development thesis, then take the software engineering classes *BEFORE* completing the project. The object-oriented software engineering paradigm taught at AFIT forces more up-front design before coding is started. This thesis would have benefited from better system design effort at the start of the project.

The other lesson is to start the actual thesis writing while the work is in progress. It is painful to recall all the work done at the early states of the project, and key development details may not be lost! It also makes the writing part of the thesis easier.

Appendix A. *List of Acronyms*

AFIT	Air Force Institute of Technology
AGC	Automatic Gain Control
BPS	Bits-Per-Second
BPSK	Binary Phase-Shift-Key
CPM/P	Command Post Modem/Processor
CPU	Central Processing Unit
DOD	Department of Defense
DSCS	Defense Satellite Communications System
EAM	Emergency Action Message
EHF	Extremely High Frequency
FET	Field Effect Transistor
FTCU	Frequency Terminal and Control Unit
GPM	General Purpose Modem
IF	Intermediate Frequency
IRIG	Inter-Range Instrumentation Group
IRQ	Interrupt Request
LNA	Low Noise Amplifier
LSB	Least Significant Bit (or Byte)
MSB	Most Significant Bit (or Byte)
PC	Personal Computer
PIO	Parallel Input/Output
PPS	Pulse-Per-Second
PRN	Pseudo-Random noise
PSK	Phase-Shift-Key
QPSK	Quadrature Phase-Shift-Key
RF	Radio Frequency
SABR	Stand Alone Beacon Receiver
SATCOM	Satellite Communications
SCT	Single Channel Transponder
SHF	Super High Frequency
TTL	Transistor-Transistor-Logic
TT&C	Telemetry, Tracking and Control
UART	Universal Asynchronous Receiver Transmitter
UHF	Ultra High Frequency
UTC	Universal Time Coordinates

```
/* ****************************************************
*   File : asc30.c
*   by Jim Coppola
*   01 Feb 92
*   updated on 23 Jul 92
*   by Jim Coppola
*   version 2.0
*
*
*   Description: Functions which implement the interface to the
* AN/ASC-30. Uses the PIO card installed in a PC. See PIO.h for
* the address and interrupt used to interface to the card. Program
* gets the bits from the ASC-30 interface, frames them up into the
* global array data for display, then displays the clock and data
* using the screen.c routines. Program quits when restart is set to
* TRUE.
*
*   Global Variables Used    : update, restart, data, index
*   Global Variables Changed : update, data, index
*
****************************************************** */

#include <dos.h>
#include <stdio.h>
#include "system.h"
#include "pio.h"

int previous = 0;
int diff = FALSE;
int invert = FALSE;
int bits[8];
int notice_up = FALSE;

/*****function to get next bit from the PIO interface*****/
int get_next()
{
  int ret, temp;

  ret = getbit() & 0x01;
  if(timed_out(parallel_timer)) {  /*****put notice up of no data*****/
    if(!notice_up){
      my_notice("No Data from ASC-30!!!");
      notice_up = TRUE;
      update = FALSE;
    }
    else {
      if(restart && notice_up) {
close_window();
notice_up = FALSE;
update = TRUE;
```

```
      }
    }
  }
  else{
    if(notice_up){
      close_window();
      notice_up = FALSE;
      update = TRUE;
    }
  }

  if(invert)  /****if asc30 locked inverted, invert data*****/
  {
    if(ret == 1)
      ret = 0;
    else
      ret = 1;
  }
  if(diff)  /*****if asc30 jumpered to put out differencial data*****/
  {
    temp = ret;
    if(previous == temp)
      ret = 1;
    else
      ret = 0;
    previous = temp;
  }
  return(ret);
}

/*****debug get next bit routine. Used to debug proglems with interface****/
int debug_next(void)
{
  int ret, temp;

  ret = getbit();
  ret = ret & 0x01;
  return ret;
}

/*****actual debug routine.  Used to watch for the bit sequences
as defined by the sct spec*****/
void debug()
{
  int found,next,count,data_in[8],i1,sequence;
  char temp;

  found = FALSE;
  count = 0;
  sequence = 24;
```

```
  restore_keyboard(); /*****take my keyboard isr out of way*****/
  while(!found)
  {

    for(i1=0;i1<8;i1++)
    {
      data_in[i1] = debug_next();
      if(i1 == 0)
printf("%d",data_in[i1]);
    }
    count++;
    if(count > sequence)
    {
      count = 0;
      printf("\n");
    }

    if(bioskey(1))
    {
      temp = bioskey(0);
      if(temp == 'q')
finish();
      if(temp == 'g')
found = TRUE;
      if(temp == 's')
      {
if(sequence < 30)
  sequence++;
sequence++;
      }
      else
      {
printf("**shift**\n");
temp = debug_next();
      }
    }
  }
  intercept_keyboard();
}

/*****function to synchronize the code to the eighth bit in
input ******/
void find_sync(void)
{
  int i1,found,shift_count,hit_count,data_in[8],temp,sync;

  found = FALSE;
  shift_count=hit_count = 0;
  sync = 0;

  while(!found)
```

```
  {
    if(restart) /*****if restart, get out now*****/
      break;
    for(i1=0;i1<8;i1++)
      data_in[i1] = get_next();

    if(data_in[0] == sync)
      hit_count++;
    else
    {
      hit_count = 0;
      shift_count++;
      if(shift_count >= 8)
      {
if(sync == 0)
{
  sync = 1;
  shift_count = 0;
}
else {
  /*****check for timeout here*****/
  sync = 0;
  shift_count = 0;
}
      }
      temp = get_next();
      temp++;  /*****just to get rid of compiler warning*****/
    }
    if(hit_count > 32)  /*****if 32 hits, we have found it!!*/
      found = TRUE;
  }
  if(sync == 1)
    invert = TRUE;
  else
    invert = FALSE;
}

/*****function to load 8 bits from the pio.  Function assumes the find
sync routine has put the sync bit as bit 0 in the sequence*****/
void load_data()
{
  int i1;

  for(i1=0;i1<8;i1++) {
    if(restart)
      break;
    bits[i1] = get_next();
  }
}

/*****function to search for message sync.  Assumes bit sync has occured.
```

```
Looks for 5 ones in a row from sequences 2 and 5 as defined by the
SCT spec*****/
int ones_search()
{
/*****need to include some sort of resync try after 8 seconds ****/
  int found, counter;

  found = FALSE;
  counter = 0;

  load_data();
  load_data();
  while(!found)
  {
    if(restart)
      break;
    load_data();
    if(bits[0] != 0)
      return FALSE;
    if((bits[1] == 1) && (bits[5] == 1))
      counter++;
    else
      counter = 0;

    if(counter > 4)
      found = TRUE;
  }
  return TRUE;
}

/*****function when system is in sync; loads the 100 bits*****/
int load_bits()
{
  int i1, error, synched, frame, framesync;
  int frame_bits[6];

  error = index = frame = 0;
  synched = TRUE;

  while(synched)
  {
    if(restart)
      break;
    data[index] = bits[7] ^ bits[3];
    index ++;
    if(index == 49)     /*****have clock data, display it*****/
      display_clock();
    if(index >= 100)
    {
      display_data(); /*****have data, display it*****/
      index = 0;
```

```
    }

    load_data();
    frame++;
    if(frame > 194)   /*****code to check frame sync.  Not implemented yet*/
      frame_bits[frame-194] = bits[1];

    load_data();
    frame++;
    if(frame > 194) {
      frame_bits[frame-194] = bits[1];
      if(frame >= 199) {
frame = 0;
framesync = 0;
for(i1=0;i1<5;i1++) {
  if(!frame_bits[i1])
    framesync++;
}
if(framesync > 1)
  synched = TRUE; /*****changed to get to run*****/
      }
    }

    if(bits[0] != 0)  /*****only sync check here. No message sync check****/
    {
      error++;
if(error > 1)
  synched = FALSE;
    }
    else
      error = 0;
  }
  return TRUE;
}

/*****procdure to capture actual beacon data to disk.  Used for ComQuest
to debug the GPM.  Left for future use if needed*****/
void capture_data()
{
  int dat[20000],i1, char_in;
  FILE *fp;
  fp = fopen("beacon.dat","w");  /*****NOTE: fixed file name!!!!!*****/

  for(i1=0;i1<20000;i1++)
  {
    char_in = get_next();
//    printf("%d", char_in);  /***commented out for proper disk operation*/
    dat[i1] = char_in;
  }

  for(i1=0;i1<20000;i1++)
```

```
  {
    fprintf(fp,"%d", dat[i1]);
  }
  fclose(fp);
}

/*****main routine in this file.  Sets up package variables and tracks
progress in synchronization process.  Displays user interfaces.  This
routine is called from main when the receiver selected is the ASC-30*****/
void asc30()
{
  int go,error;
  setportin();
  initport();
  go = TRUE;
  while(go)
  {
    invert = FALSE;
    if(restart)       /*****if restart, return to main*****/
      break;
    update = FALSE;
    notice_up = FALSE;
//    debug();        /*****if you like to degub, uncomment*****/

    my_notice("Looking for Sync Bit");
    find_sync();
    close_window();
    my_notice("Looking for Message Sync");
    go = ones_search();
    close_window();
    update = TRUE;
    go = load_bits();
  }
}
```

```
/*****************************************************
*File   : control.c
*By     : Jim Coppola
*       : 24 Sep 92
*       : Version 1.0
*Update:
*By     :
*
*Description:   Fuctions to talk to the beacon receivers
*
*   Global Variables Used    : com
*   Global Variables Changed : None
*
*****************************************************/

#include <conio.h>
#include <stdio.h>
#include "system.h"
#include "keys.h"
#include "port.h"


int wait_for_prompt(void)
{
  char char_in;
  unsigned port;

  port = gpm_control_port;

  for(;;) {
    char_in = get_char(port);
    putch(char_in);
    if(char_in == '>')
      return TRUE;
    if(timed_out(port))
      return FALSE;
  }
}

void control_out(char *string_in)
{
  char char_out, char_in;
  unsigned port = gpm_control_port;
  int temp;

  char_out = *(string_in);
  while(char_out != 0) {
    put_char(port,char_out);
    string_in++;
    char_in = get_char(port);
```

```
    putch(char_in);
    char_out = *(string_in);
  }
  char_out = CR;
  put_char(port,char_out);
  temp = wait_for_prompt();
  temp++;  /*just to get rid of compiler warning*/
}

int gpm_start(void)
{
  int i1, found, test;
  unsigned com;
  char char_out, char_in;
  char command1[] = {"RM 0"};
  char command2[] = {"RR 3200"};
  char command3[] = {"DD 0"};
  char command4[] = {"SD 0"};
  char command5[] = {"RC 0"};
  char command6[] = {"DMD 4"};
  char command7[] = {"GRC 0"};
  char command8[] = {"GMD 4"};

  com = gpm_control_port;
  openport(com,1200,7,1,0);

  open_window(10,5,70,20,WHITE,BLACK,2,0);
  gotoxy(1,1);

  found = FALSE;
  char_out = CR;
  for(i1=0;i1<3;i1++) {
    put_char(com, char_out);
    test = wait_for_prompt();
    if(test) {
      found = TRUE;
      i1 = 4;
    }
  }
  if(!found) {
    error_message("Bad Link To GPM");
    close_window();
    return FALSE;
  }
  clear_serial_queue(com);
  char_out = CR;
  put_char(com,char_out);
  test = wait_for_prompt();
  control_out(command1);
  control_out(command2);
  control_out(command3);
```

```
  control_out(command4);
  control_out(command5);
  control_out(command6);
  control_out(command7);
  control_out(command8);
  set_timer(0,3);
  while(!timed_out(0));
  close_window();
  return TRUE;
}

void gpm_control(void)
{
  unsigned com;
  char char_in, char_out;

  com = gpm_control_port;
  openport(com,1200,7,1,0);

  open_window(15,2,65,4,RED,LIGHTGRAY,1,0);
  cputs("       Hit <ESC> To Quit GPM Command Window");
  open_window(10,5,70,20,WHITE,BLACK,2,0);
  gotoxy(1,1);
  clear_serial_queue(com);
  char_out = CR;
  put_char(com,char_out);
  char_out = 0;
  while(char_out != ESC) {
    if(char_rdy(com)) {
      char_in = get_char(com);
      putch(char_in);
    }
    if(bioskey(1)) {
      char_out = bioskey(0);
      put_char(com,char_out);
    }
  }
  close_window();
  close_window();
}

void setafreq(void)
{
  unsigned com;

  com = beacon_port;

  put_char(com,0x24);
  put_char(com,0x00);
  put_char(com,0x10);
  put_char(com,0x1B);
```

```
}

void setbfreq(void)
{
  unsigned com;
  char char_out;

  com = beacon_port;

  clear_xmit_queue(com);
  put_char(com,0x42);
  put_char(com,0x1E);
  put_char(com,0x6E);
  put_char(com,0x1B);
}

/*****procedure to compute and send the frequency offset to
  the raytheon beacon receiver.  Code was converted from the
  Pascal code for the beacon receiver. *****/
void offset_compute(int offset)
{
  int freq_op, local_offset;
  int byte1,byte2,byte3,byte4,byte5;
  float foffset, k, temp_ls, temp_ms;
  unsigned com, offset_low, offset_high;

  com = beacon_port;

  k = 781.25; /** see pascal source code**/
  if(offset > 0)
    freq_op = 0x00;
  else
    freq_op = 0x10;

  local_offset = abs(offset)/64;
  foffset = (float) local_offset;
  offset_high = (unsigned) (foffset/k);
  temp_ms = foffset/k;
  temp_ls = temp_ms - (float) offset_high;
  offset_low = (unsigned) (temp_ls * 65536);

  byte1 = freq_op;
  byte2 = offset_low & 0xFF;
  byte3 = offset_low >> 8;
  byte4 = offset_high & 0xFF;
  byte5 = offset_high >> 8;

  put_char(com,byte1);
  put_char(com,byte2);
  put_char(com,byte3);
  put_char(com,byte4);
```

```
  put_char(com,byte5);
}

void ray_start(void)
{
  int i1, notice_up, quit;
  unsigned com;
  char char_in, char_out, key_in;
  char complete[] = {"COMPLETE"};

  com = beacon_port;
  openport(com,9600,8,0,1);

  open_window(13,2,67,4,RED,LIGHTGRAY,1,0);
  cputs("Hit <ESC> To Quit Raytheon Stand Alone Start Window");
  open_window(1,5,80,23,WHITE,BLACK,2,0);
  gotoxy(1,1);
  clear_serial_queue(com);
  my_notice("Please Press RESET On Raytheon Stand Alone");
  notice_up = TRUE;
  char_out = 0;
  i1 = 0;
  while((char_out != ESC)) {
    if(char_rdy(com)) {
      if(notice_up) {
close_window();
notice_up = FALSE;
      }
      char_in = get_char(com);
      putch(char_in);
      if(char_in == complete[i1]) {
i1++;
if(i1 >= 7)
  break;
      }
      else
i1 = 0;
    }
    if(bioskey(1)) {
      char_out = bioskey(0);
    }
  }
  if(notice_up)
    close_window();
  close_window();
  if(char_out == ESC) {
    close_window();
    return;
  }
  open_window(10,5,70,15,WHITE,BLACK,2,0);
  cputs("Select Beacon \r\n");
```

```
  cputs("A For 7600 MHz \r\n");
  cputs("B For 7604.705 MHz \r\n");
  cputs("Beacon?? ");
  quit = FALSE;
  while(!quit) {
    key_in = bioskey(0);
    if(key_in == ESC)
      quit = TRUE;
    if(tolower(key_in) == 'a') {
      setafreq();
      quit = TRUE;
    }
    if(tolower(key_in) == 'b') {
      setbfreq();
      quit = TRUE;
    }
  }
  offset_compute(0);
  close_window();
  close_window();
}

void ray_control(void)
{
  char hzstring[10];
  char *pointer;
  int offset;

  open_window(10,2,70,4,RED,LIGHTGRAY,1,0);
  cputs("        Hit <ESC> To Quit Frequency Offset Window");
  open_window(10,5,70,10,WHITE,BLACK,2,0);
  cputs("Enter Frequency Offset In Hz : ");
  hzstring[0] = 6;
  my_cgets(hzstring);
  pointer = hzstring + 2;
  offset = atoi(pointer);
  offset_compute(offset);
  close_window();
  close_window();
}
```

```
/****************************************************
*File  : cursor.h
*By    : Jim Coppola
*      : 17 Aug 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Header file for cursor functions.
*
*
****************************************************/

void show_cursor(void);
void hide_cursor(void);
void block_cursor(void);
void default_cursor(void);
void savecursor(void);
void restorecursor(void);
void normalcursor(void);


/****************************************************
*File  : cursor.c
*By    : Jim Coppola
*      : 17 Aug 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Cursor manipulation routines.  Performs cursor tricks.
*  Only hidecursor() and show_cursor are used by sct program
*
*   Global Variables Used    : None
*   Global Variables Changed : None
*
*
****************************************************/

#include <dos.h>
#include <conio.h>
#include "cursor.h"
#include "interupt.h"

static void interrupt far (*oldkb)(void);
static void interrupt far newkb(IREGS);
static void invert_block(void);

/*****kekyboard bios (ox16) functions*****/
#define READKB 0
```

```
#define KBSTAT 1

/*****video BIOS (0x10) functions*****/
#define SETCURSORTYPE 1
#define SETCURSOR     2
#define READCURSOR    3
#define READATTRCHAR  8
#define WRITEATTRCHAR 9
#define HIDECURSOR 0X20

static int cursorpos;
static int cursorshape;

/*****get cursor shape and position*****/
static void getcursor(void)
{
  _AH = READCURSOR;
  _BH = 0;
  geninterrupt(VIDEO);
}

/*****saave current cursor config*****/
void savecursor(void)
{
  getcursor();
  cursorshape = _CX;
  cursorpos = _DX;
}

/*****restore saved cursor config*****/
void restorecursor(void)
{
  _AH = SETCURSOR;
  _BH = 0;
  _DX = cursorpos;
  geninterrupt(VIDEO);
  _AH = SETCURSORTYPE;
  _CX = cursorshape;
  geninterrupt(VIDEO);
}

/*****make a normal cursor*****/
void normalcursor(void)
{
  _AH = SETCURSORTYPE;
  _CX = 0x0607;
  geninterrupt(VIDEO);
}

/*****hide the cursor*****/
void hidecursor(void)
```

```
{
  getcursor();
  _CH |= HIDECURSOR;
  _AH = SETCURSORTYPE;
  geninterrupt(VIDEO);
}

/*****show the cursor*****/
void show_cursor(void)
{
  getcursor();
  _CH &= ~HIDECURSOR;
  _AH = SETCURSORTYPE;
  geninterrupt(VIDEO);
}

/*****make a block cursor*****/
void block_cursor(void)
{
  hidecursor();
  oldkb = getvect(KEYBOARD);
  setvect(KEYBOARD,newkb);
}

/*****return to default blinking cursor*****/
void default_cursor(void)
{
  show_cursor();
  setvect(KEYBOARD,oldkb);
}

/*****interrupt function to display and clear the block cursor*****/
void interrupt far newkb(IREGS ir)
{
  static int cset = 0;
  static char kfunc;

  kfunc = _AH;  /***bios function requested***/
  if(kfunc == READKB || kfunc == KBSTAT) {
    if(cset == 0) {
      cset = 1;
      invert_block();
    }
  }
  _AX = ir.ax;
  (*oldkb)();
  ir.ax = _AX;
  ir.fl = _FLAGS;
  if(kfunc == READKB && cset == 1) {
    cset = 0;
    invert_block();
```

```
  }
}

/*****invert the video attributes at the current cursor pos*****/
static void invert_block(void)
{
  static int c, at;
  /**read attribute and char at the cursor pos*****/
  _BH = 0;
  _AH = READATTRCHAR;
  geninterrupt(VIDEO);
  c=_AX;
  /****swap back RGB With fg RGB*****/
  at = (c >> 8) & 0xFF;
  at = ((at >> 4) & 7) | ((at << 4) & 0x70) | (at & 0x88);
  /***write attribute and char at the cursor*****/
  _CS = 1; /**one char to write**/
  _BH = 0; /**page zero**/
  _BL = at; /**new attribute**/
  _AL = c & 0xFF;   /**character**/
  _AH = WRITEATTRCHAR;
  geninterrupt(VIDEO);
}
```

```
/*********************************************************
*   File : gpm.c
*   by Jim Coppola
*   18 Jul 92
*   version 1.0
*
*Description:  Contains functions to interface to the GPM beacon port,
*receive the beacon data, then call update routines to decode and display
*the SCT configuration.
*
*   Global Variables Used    : update, restart, data, index, remote
*   Global Variables Changed : update, data, index, remote
*
*
******************************************************/

#include <dos.h>
#include <stdio.h>
#include "system.h"
#include "port.h"
#include "window.h"

int gpm_notice_up;

/*****function to strip the ascii bits out of the character sent
******stripping out four bits at a time for GPM*****/
int stripbits(int charin)
{
  int i2,temp;
  if(REMOTE_SLAVE)
    charin = charin & 0x7F;
  if(charin < 0x3A)
    charin = charin - 48;
  else
    charin = charin - 55;

  for(i2=0;i2<4;i2++) {
    temp = charin & 0x08;
    if(temp == 0)
      data[index] = 0;
    else
      data[index] = 1;

    charin = charin << 1;
    index++;
    if(index == 49)
      display_clock();
    if(index >= 100)
      return TRUE;
    }
```

```
    return FALSE;
}

/*****function to get character from gpm.  Checks for timeout condition,
posts warning message if timed out.  Notice removed once data appears*****/
int local_get_char()
{
  unsigned com;
  int char_in, test;
  com = beacon_port;
  char_in = get_char(com);
  test = timed_out(com);
  if(test) {
    if(!gpm_notice_up && !restart){
      my_notice("No Data from GPM!!!");
      gpm_notice_up = TRUE;
      update = FALSE;
    }
    else {
      if(restart && gpm_notice_up) {
close_window();
gpm_notice_up = FALSE;
update = TRUE;
      }
    }
  }
  else{
    if(gpm_notice_up){
      close_window();
      gpm_notice_up = FALSE;
      update = TRUE;
    }
  }
  return (char_in);
}

/*****main gpm procedure.  Opens port, waits for line feed, then gets
characters and call strip bits to parse out the data.  If remote master
or remote slave, the satellite name is handshaked across after the data
is displayed. *****/
void gpm()
{
  unsigned com;
  int char_in, go, finished, j1, i1, i2, update_labels;
  char print_buffer[4096];
  char quit[] = "ATHO";

  com = beacon_port;
  j1 = 0;
  update_labels = FALSE;
```

```
/*****Open port if not remote slave.  If remote slave, rs function opens
the port to talk to the modem*****/
  if(!REMOTE_SLAVE)
    openport(com,9600,7,1,0);

  go = TRUE;

  while(go)
  {
  /*****restart means get back to main for something ASAP*****/
    if(restart)
      return;

    gpm_notice_up = FALSE;

    char_in = 0;
    /*****trap on line feed*****/
    while(char_in != LF )
    {
      if(restart)
return;
      char_in = local_get_char();
    }
    /*****skip line feeds*****/
    char_in = local_get_char();
    char_in = local_get_char();
    /*****get bits of telemetry data*****/
    index = 0;
    while(index < 100)
    {
      if(restart)
return;
      char_in = local_get_char();

      finished = stripbits(char_in);
      /*****if 100 bits, then break to display.  Clock is displayed in
      stripbits when the 48 bits have been obtained*****/
      if(finished)
break;
    }
    if(restart)
      return;
    display_data();
    if(update_labels) {
      display_sat();
      update_labels = FALSE;
    }
    /*****check remoting items*****/
    if(REMOTE_SLAVE) {
      char_in = local_get_char();
      char_in = local_get_char();
```

```
      if(char_in == SOH) {
for(i1=0;i1<3;i1++) {
  char_in = local_get_char();
  if(char_in != sat[i1]) {
    sat[i1] = char_in;
    update_labels = TRUE;
  }
}
      }
    }
    if(REMOTE) {
      char_in = local_get_char();
      while(!xmit_buffer_empty(remote_port))
xmit_char(remote_port);
      put_char(remote_port,SOH);
      for(i1=0;i1<3;i1++)
put_char(remote_port,sat[i1]);

      if(char_rdy(remote_port)) {
char_in = get_char(remote_port);
if(char_in == quit[j1]) {
  j1++;
  while(char_rdy(remote_port)) {
    char_in = get_char(remote_port);
    if(char_in == quit[j1]) {
      j1++;
      if(j1 >= 3)
krm();
    }
    else {
      j1 = 0;
      break;
    }
  }
}
else
  j1 = 0;
      }
    }
  }
}
```

```
/******************************************************
*File  : init.c
*By    : Jim Coppola
*      : 19 Jul 92
*      : Version 1.0
*Update: 29 Jul 92
*By    : Jim Coppola
*
*Description: Contains functions to get receiver, set up global
*variables used by the system, and set up the initial screen.
*
*   Global Variables Used    : update, restart, data[], old[], FGCOLOR,
*PRINTER_UP, DISK_LOG_UP, B_BIRD, FIRST, IRIG_UP, receiver, options,
*quitnow, sat[], beacon_port, gpm_control_port, remote_port, REMOTE,
*dos_timer_count, DOS_TIMER_UP, printnow, disknow, capturenow, print_port
*
*   Global Variables Changed    : update, restart, data[], old[], FGCOLOR,
*PRINTER_UP, DISK_LOG_UP, B_BIRD, FIRST, IRIG_UP, receiver, options,
*quitnow, sat[], beacon_port, gpm_control_port, remote_port, REMOTE,
*dos_timer_count, DOS_TIMER_UP, printnow, disknow, capturenow, print_port

******************************************************/

#include <conio.h>
#include "system.h"
#include "window.h"
#include "timer.h"

/*****time out function - not used in this system*****/
static void time_out(void)
{
}

/*****Used to debug decode and display software*****/
void test_display_load(void)
{
  int i1;
  for(i1=0;i1<100;i1++)
    data[i1] = 0;

  data[51] = 1;

  data[60] = 1;
}

/*****function to initialize global variables and misc*****/
void init_data(void)
{
  int i1;
```

```
  for(i1=0;i1<100;i1++)
    old[i1] = 0xFF;

  FGCOLOR = YELLOW;
  BGCOLOR = BLUE;

  PRINTER_UP = DISK_LOG_UP = FALSE;
  options = quitnow = FALSE;
  B_BIRD = FIRST = TRUE;
  restart = IRIG_UP = FALSE;
  receiver = 0;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '4';
  /*****com port assignments*****/
  beacon_port = 1;
  gpm_control_port = 2;
  remote_port = 2;
  REMOTE = FALSE;

  /*****set up dos timer stuff*****/
  dos_timer_count = 17;
  DOS_TIMER_UP = TRUE;

/***** set up printing stuff *****/
  printnow = disknow = capturenow = FALSE;
  print_port = 0;

//  test_display_load();
}

/*****main function in file.  Calls init data to initialize global
variables.  Also calls options() to get the user's initial inputs.  Sets
up the initial display screen*****/
void init(void)
{
  int status;

  textbackground(GREEN);
  textcolor(WHITE);
  clrscr();
  init_data();
  status = initialize_printer();
  if(!status)
    PRINTER_UP = FALSE;
  else
    PRINTER_UP = TRUE;
  update = FALSE;
  intercept_timer(time_out);
  set_default_options();
  /*****START_REMOTE set in command line arguments evaluated in main()*****/
```

```
  if(START_REMOTE)
    rs();
  INITIAL = TRUE;
  if(!REMOTE_SLAVE)
    get_options();
  INITIAL = FALSE;
  clear_keys();
  intercept_keyboard();

  open_window(1,1,80,25,FGCOLOR,BGCOLOR,FALSE,0);
  display_labels();
  display_system();
  update = TRUE;
  hidecursor();
}
```

```
/*****************************************************
*File  : interrupt.h
*By    : Jim Coppola
*      : 19 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description:Header file for the interrupt vectors that may
* be used by various interrupt handlers as they are developed.
*
*
*
*
*
*****************************************************/

/*****the interrupt function registers*****/

typedef struct
{
  int bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,fl;
} IREGS;

#define VIDEO  0x10
#define DOS 0x21
#define KYBRD 9
#define KEYBOARD 0x16
#define TIMER 8

#define KBDATA 0x60
#define KBCONTROL 0x61
#define KBSTATUS 0x64

#define ZEROFLAG 0x40
```

```
/*****************************************************
*File  : irig.h
*By    : Jim Coppola
*      : 02 Sep 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Header file for irig code to talk to irig card.
*
*
*****************************************************/

/*****function declarations*****/
int start_irig();

/*****irig register addresses*****/
const INT_REGISTER = 0x0F;
const CMDBYT       = 0x0E;
const PDLAY_MSB    = 0x0D;
const PDLAY_LSB    = 0x0C;
const HEARTBT_MSB  = 0x0B;
const HEARTBT_LSB  = 0x0A;
const MASKS        = 0x09;
const TCODE        = 0x08;

const STATUS       = 0x07;
const DAT06        = 0x06;
const DAT05        = 0x05;
const DAT04        = 0x04;
const DAT03        = 0x03;
const DAT02        = 0x02;
const DAT01        = 0x01;
const DAT00        = 0x00; /* I/O data block assignments offset
       from base address */
const DELAY = 100;
const TM_OUT_VAL = 30000;

/*****base address of irig card.  Using prn2 interrupt*****/
const BASE = 0x320;  /***hex base address of irig card***/

/*****irig interrupt values : using prn2 interrupt*****/
#define IRIGENABLE      0xDF
#define IRIGDISABLE     0x20
#define IRIGINT         0x0D
#define PIC01 0x21 /*8259  interrupt controller*/
#define PIC00 0x20 /*8259  interrupt controller*/
#define EOI 0x20 /*end of interrupt*/

/*****************************************************
```

```
*File   : irig.c
*By     : Jim Coppola
*       : 02 Sep 92
*       : Version 1.0
*Update:
*By     :
*
*Description:   Functions to implement irig interface
*
*   Global Variables Used    :
*   Global Variables Changed : IRIG_UP, mil_date[], sys_time[],
*INTERUPT_UP
*****************************************************/
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include "system.h"
#include "irig.h"


void tim_out(void);
void tim_delay();
static void (interrupt far *oldirigint)(void);
static void interrupt far newirigint(void);

char old_date[] = {"000"};

/* ****post irig date in military format**** */
/* ****first, convert julian date to dos time format**** */
void post_date(int date_in)
{
  int i1, month, day, year, month_offset;
  char months[] = {"JanFebMarAprMayJunJulAugSepOctNovDec"};
  char temp[5];
  float d, m, y, j, jdate_in,daytemp;
  struct date dos_date;

  getdate(&dos_date);
  daytemp = datetojulian(1, 1, dos_date.da_year);
  daytemp++; /*just to get rid of compiler warning!!*/
  jdate_in = irig_first_day - 1.0 + (float) date_in;

  j = (long) jdate_in - 1721119.0;
  y = (long) ((4*j-1)/146097.0);
  j = (long) ((4*j-1)-(146097.0*y));
  d = (long) (j/4);
  j = (long) ((4*d+3)/1461);
  d = (long) ((4*d+3)-(1461*j));
  d = (long) ((d+4)/4);
  m = (long) ((5*d-3)/153);
```

```
  d = (long) ((5*d-3)-(153*m));
  d = (long) ((d+5)/5);
  y = (long) (100*y+j);

  day = (int) d;
  month = (int) m;
  year = (int) y;
  if(month < 10)
    month = month + 3;
  else {
    month = month = 9;
    year = year + 1;
  }

  itoa(day,temp,10);
  if(day < 10) {
    mil_date[0] = '0';
    mil_date[1] = temp[0];
  }
  else {
    mil_date[0] = temp[0];
    mil_date[1] = temp[1];
  }
  mil_date[2] = ' ';
  month_offset = (month - 1) * 3;
  for(i1=0;i1<3;i1++)
    mil_date[i1+3] = months[i1 + month_offset];
  mil_date[6] = ' ';
  itoa(year,temp,10);
  mil_date[7] = temp[2];
  mil_date[8] = temp[3];
  mil_date[9] = 0; /**null terminator**/
}

/*****interrupt service routine for 1pps irig*****/
static void interrupt far newirigint(void)
{
  int i, j, i1, data, rdyflg, timebyte, post, irig_jtime;
  long int offset;
  unsigned int timer;
  char *pointer = sys_time;

  INTERRUPT_UP = TRUE;
  sys_time[0] = inportb(BASE + INT_REGISTER); /* clear irq */
  outp(PIC00,EOI);

  timer = TM_OUT_VAL;
  outportb(BASE+CMDBYT,0x80);         /* request time */
  do  {
    tim_delay();                      /* slow polling rate to < 500 Khz */
    rdyflg = inportb(BASE+CMDBYT);
```

```
  }
  while ((rdyflg != 0) && --timer);  /* wait for ready flag of 0 */

  if(!timer)
    tim_out();

  if(IRIG_UP) {
    i = 0;            /* begin loading time into time[j] array */
    j = 12;

    sys_time[j--] = 0; /*****null terminator */
    sys_time[j--] = (inportb(BASE+i) & 0x0F) | 0x30; /* get seconds units */
    sys_time[j--] = ((inportb(BASE+i++) & 0xF0) >> 4) | 0x30;  /* get seconds
      tens  */
    diff_time = atoi(pointer+10);
    sys_time[j--] = ' ';
    sys_time[j--] = (inportb(BASE+i) & 0x0F) | 0x30;  /* get minutes units */
    sys_time[j--] = ((inportb(BASE+i++) & 0xF0) >> 4) | 0x30;  /* get minutes
      tens  */
    sys_time[j--] = ' ';
    sys_time[j--] = (inportb(BASE+i) & 0x0F) | 0x30;  /* get hours units */
    sys_time[j--] = ((inportb(BASE+i++) & 0xF0) >> 4) | 0x30;  /* get hours
      tens  */
    sys_time[j--] = '/';
    sys_time[j--] = (inportb(BASE+i) & 0x0F) | 0x30;  /* get days units */
    sys_time[j--] = ((inportb(BASE+i++) & 0xF0) >> 4) | 0x30;  /* get days
      tens  */
    sys_time[j]   = ((inportb(BASE+i) & 0xC0) >> 6) | 0x30;    /* get days
      hundreds */
    post = FALSE;
    for(i1=j;i1<j+3;i1++) {
      if(old_date[i1] != sys_time[i1]) {
old_date[i1] = sys_time[i1];
post = TRUE;
      }
    }
    if(post) {
      irig_jtime = atoi(old_date);
      post_date(irig_jtime);
      post = FALSE;
    }
    display_time();
  }
}

/*****initialize irig card port and set up interrupt vector(s)*****/
void intercept_irig()
{
/*****set up interrupt vector*****/
  outp(PIC01, (inp(PIC01) | IRIGDISABLE));
  oldirigint = getvect(IRIGINT);
```

```
  setvect(IRIGINT, newirigint);
  outp(PIC01, (inp(PIC01) & IRIGENABLE));
}


/*****restore the interrupt vector*****/

void restore_irig(void)
{
  int rdyflg;
  unsigned int timer=TM_OUT_VAL;

  IRIG_UP = FALSE;
//  if(oldirigint)
  setvect(IRIGINT, oldirigint);

  outp(PIC01, (inp(PIC01) | IRIGDISABLE));

  outportb(BASE+MASKS,0x0);
  outportb(BASE+CMDBYT,0x90);   /* initialize 630AT to above values */
       /* **** INITIALIZE COMMAND MUST FOLLOW ALL     *** */
       /* **** UPDATES TO TIMECODE AND MASKS REGISTER *** */

  do           /* Poll for 'Initialize' complete ready flag of 0x10 */
  {
    tim_delay();              /* slow polling rate to < 500 Khz */
    rdyflg = inportb(BASE+CMDBYT);
  }
  while ((rdyflg != 0x10) && --timer);    /* wait for ready flag of 0x10 */

  if(!timer)
    tim_out();

}

/*****initialize irig card, call isr replacement routine*****/
int start_irig()
{
  int i1, rdyflg, ret;
  unsigned int timer=TM_OUT_VAL;
  intercept_irig();
  INTERRUPT_UP = FALSE;
  for(i1=0;i1<3;i1++)
    old_date[i1] = '0';

  outportb(BASE+STATUS,0x00);
  tim_delay();
  outportb(BASE+TCODE,0x42);   /***** select detect of incoming
    IRIG B modulated and
    time codes
    init mode = normal / modulated time code select
```

```
    Note: for autodetect mod/DC
    bit must be zero*****/

  outportb(BASE+CMDBYT,0x90);   /* initialize 630AT to above values */
       /* **** INITIALIZE COMMAND MUST FOLLOW ALL       *** */
       /* **** UPDATES TO TIMECODE AND MASKS REGISTER *** */

  do           /* Poll for 'Initialize' complete ready flag of 0x10 */
  {
    tim_delay();              /* slow polling rate to < 500 Khz */
    rdyflg = inportb(BASE+CMDBYT);
  }
  while ((rdyflg != 0x10) && --timer);    /* wait for ready flag of 0x10 */

  /*****initialize interrrupts*****/
  timer = TM_OUT_VAL;
  outportb(BASE+MASKS,0x88);
  outportb(BASE+CMDBYT,0x90);   /* initialize 630AT to above values */
       /* **** INITIALIZE COMMAND MUST FOLLOW ALL       *** */
       /* **** UPDATES TO TIMECODE AND MASKS REGISTER *** */

  do           /* Poll for 'Initialize' complete ready flag of 0x10 */
  {
    tim_delay();              /* slow polling rate to < 500 Khz */
    rdyflg = inportb(BASE+CMDBYT);
  }
  while ((rdyflg != 0x10) && --timer);    /* wait for ready flag of 0x10 */

  if(!timer) {
    tim_out();
    return FALSE;
  }
  else  {
    IRIG_UP = TRUE;
    return TRUE;
  }
}

/*****delay to slow card polling rate to acceptable limit*****/
void tim_delay()
{
     int ii;

     for (ii = DELAY; ii != 0; ii--);
}

/*****error routine when IRIG card doesn't respond*****/
void tim_out()
{
  setvect(IRIGINT, oldirigint);
  outp(PIC01, (inp(PIC01) | IRIGDISABLE));
```

```
  error_message("IRIG Card Not responding!!!");
  IRIG_UP = FALSE;
}
```

```
/*****************************************************************
*File   : keys.h
*By     : Jim Coppola
*       : 29 Jul 92
*       : Version 1.0
*Update:
*By     :
*
*Description: Header file for keyboard definitions.  Routines taken
* from "Extending Trubo C professional by Al Stevens.
*
*
******************************************************************/

#define RUBOUT  8
#define BELL    7
#define ESC    27

#define F1  187
#define F2  188
#define F3  189
#define F4  190
#define F5  191
#define F6  192
#define F7  193
#define F8  194
#define F9  195
#define F10 196

#define CTRL_F1  222
#define CTRL_F2  223
#define CTRL_F3  224
#define CTRL_F4  225
#define CTRL_F5  226
#define CTRL_F6  227
#define CTRL_F7  228
#define CTRL_F8  229
#define CTRL_F9  230
#define CTRL_F10 231

#define ALT_F1  232
#define ALT_F2  233
#define ALT_F3  234
#define ALT_F4  235
#define ALT_F5  236
#define ALT_F6  237
#define ALT_F7  238
#define ALT_F8  239
#define ALT_F9  240
#define ALT_F10 241
```

```
#define HOME   199
#define UP      200
#define PGUP 201
#define BS 203
#define FIVE 204
#define FWD 205
#define END 207
#define DN 208
#define PGDN 209
#define INS 210
#define DEL 211

#define CTRL_HOME  247
#define CTRL_UP    141
#define CTRL_PGUP  132
#define CTRL_BS    243
#define CTRL_FIVE  143
#define CTRL_FWD   244
#define CTRL_END   245
#define CTRL_DN    145
#define CTRL_PGDN  246
#define CTRL_INS   146
#define CTRL_DEL   147

int getkey(void);
void aurotype(char *);
void repeat_rate(int);
void stuffkey(int);
void clear_biosbuff(void);

extern int cgets_returnkey;

/*****************************************************
*File   : keyboard.c
*By     : Jim Coppola
*       : 17 Aug 92
*       : Version 1.0
*Update:
*By     :
*
*Description:  Functions which interface with the keyboard
*
*   Global Variables Used    :
*   Global Variables Changed : quitnow, update, capturenow, restart

*****************************************************/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
```

```
#include <ctype.h>
#include "system.h"
#include "keys.h"
#include "interupt.h"

static void (interrupt far *oldkeyboard)();
static void interrupt far newkeyboard();

int cgets_returnkey;

/*****intercept the keyboard interrupt vector*****/
void intercept_keyboard(void)
{
    oldkeyboard = getvect(KYBRD);
    setvect(KYBRD, newkeyboard);
}

/*****restore keyboard interrupt vector*****/
void restore_keyboard()
{
  if(oldkeyboard != NULL)
    setvect(KYBRD, oldkeyboard);
}

/*****routine to get a key******/
int getkey(void)
{
  int c;

  while (bioskey(1) == 0)
    geninterrupt(0x28);
  if(((c=bioskey(0)) & 255) == 0)
    c = (c >> 8) | 0x80;
  return c & 255;
}

/*****routine to post sure quit notice or call finish to quit*****/
void quit1(void)
{
  if(quitnow){
    close_window();
    finish();
  }
  else {
    update = FALSE;
    quitnow = TRUE;
    my_notice("Q or Y to really quit!! ");
  }
}

/*****ISR to check keyboard*****/
```

```
static void interrupt far newkeyboard()
{
  unsigned char key_in;

  (*oldkeyboard)();

  if(bioskey(1)) {
    key_in = getkey();

    switch(key_in) {
      case 'y' : if(quitnow)
   quit1();
      break;
      case 'q' : quit1();
      break;
      case 'Y' : if(quitnow)
   quit1();
      break;
      case 'Q' : quit1();
      break;
      case F1        : restart = TRUE;
       options = TRUE;
      break;
      case F2        : restart = TRUE;
       printnow = TRUE;
      break;
      case F4        : gettext(1,1,80,25,screen_buffer);
       capturenow = TRUE;  /**for thesis screen capture**/
       restart = TRUE;
      break;
      default        : if(quitnow) {
 quitnow = FALSE;
 update = TRUE;
 close_window();
       }
    }
  }
}

/*****function to clear keyboard buffer*****/
void clear_keys(void)
{
  int key_in;

  while(bioskey(1))
    key_in = bioskey(0);
  key_in++;  /*****to get rid of compiler warning*****/
}


/*****improved cgets routine, workes nice, stays in active window*****/
```

```
char *my_cgets(char *s)
{
  char *bf = s + 2;
  int c = 0;

  while ((c = getkey()) != '\r') {
    if(c==BS)
      c = '\b';
    if(c== '\b') {
      if(bf > s+2) {
--bf;
cputs("\b \b");
      }
    }
    else if(isprint(c) && bf < s + *s + 1) {
      putch(c);
      *bf++ = c;
    }
    else if (c==ESC || c & 0x80)
      break;
    else
      putch(BELL);
  }
  *bf = '\0';
  s[1] = bf - s - 2;
  cgets_returnkey = c;
  return s+2;
}
```

```c
/*****************************************************
*File  : main.c
*By    : Jim Coppola
*      : 19 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description: main procedure here.  Also includes finished routine
*which restores all interrupts and closes ports.  Calls menu when F1
*is pressed. Checks command line argument for slave operation.
*
*
*   Global Variables Used    : update, restart, receiver, capturenow,
*printnow, disknow, PARALLEL_UP, IRIG_UP
*   Global Variables Changed : update, data, START_REMOTE, options
*first, capturenow, printnow, disknow, DISK_LOG_UP
*****************************************************/

#define MAIN
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include "system.h"

/***** function prototypes*****/
void finish(void);
void store_screen(void);

/*****required main function which controls the system*****/
void main(int argc, char *argv[])
{
  int i1, i2;
  char rem[] = {'r','e','m','o'};
  char print_buffer[4096];
  char *pointer, char_out, file_name[11];
  int sheet = 0;
  FILE *file_pointer;

  /*****check command line argument*****/
  if(argc > 1) {
    for(i1=0;i1<4;i1++) {
      if((tolower(*(argv[1]+i1))) == rem[i1]){
START_REMOTE = TRUE;
      }
      else {
START_REMOTE = FALSE;
break;
      }
```

```
    }
  }

  init();
  update = TRUE;
  for(;;) {
    switch(receiver)
    {
      case 0 : gpm();
    break;
      case 1 : raytheon();
    break;
      case 2 : asc30();
    break;
     default : notice("Program Error!!! Panic Quit");
       finish();
    }
    if(options) {
      restart = FALSE;
      restore_keyboard();
      update = FALSE;
      options = FALSE;
      get_options();
      for(i1=0;i1<100;i1++)
old[i1] = 0xFF;
      close_all_windows();
      open_window(1,1,80,25,FGCOLOR,BGCOLOR,FALSE,0);
      FIRST = TRUE;
      update = TRUE;
      display_labels();
      display_system();
      intercept_keyboard();
    }
    if(capturenow) {
      restart = FALSE;
      update = FALSE;
      capturenow = FALSE;
      restore_keyboard();
      store_screen();
      update = TRUE;
      intercept_keyboard();
    }
    if(printnow) {
      restart = FALSE;
      update = FALSE;
      printnow = FALSE;
      if(printer_ready()) {
gettext(1,1,80,25,print_buffer);
my_notice("Printing Beacon Information");
pointer = print_buffer;
for(i1=0;i1<4;i1++)
```

```
  print_char(0x0A);
print_char(0x0D);
i2 = 0;
for(i1=0;i1<1990;i1++) {
  char_out = *(pointer);
  print_char(char_out);
  i2++;
  if(i2 >= 80) {
    char_out = 0x0D;
    print_char(char_out);
    char_out = 0x0A;
    print_char(char_out);
    i2 = 0;
  }
  pointer = pointer+2;
}
sheet++;
if(sheet >= 2) {
  char_out = 0x0C;
  print_char(char_out);
  sheet = 0;
}
close_window();
char_out = 0x0D;
print_char(char_out);
      }
      else {
restore_keyboard();
error_message("Printer Error, Cannot Print!!");
intercept_keyboard();
/*PRINTER_UP flag set in printer handler*/
      }
      update = TRUE;
    }
    if(disknow) {
      restart = FALSE;
      update = FALSE;
      disknow = FALSE;
      gettext(1,1,80,25,print_buffer);
      pointer = print_buffer;
      disknow = FALSE;

      sprintf(file_name,"%c%c%c%c%c%c%c%c%c%c%c",sys_time[0],sys_time[1],
sys_time[2],sys_time[7],sys_time[8],
sys_time[10],sys_time[11],
'.','c','a','p');
      file_pointer = fopen(file_name,"wt");
      if(file_pointer != NULL) {
my_notice("Saving Beacon Information To Disk");
for(i1=0;i1<4;i1++)
  fprintf(file_pointer,"%c",0x0A);
```

```
fprintf(file_pointer,"%c",0x0D);
i2 = 0;
for(i1=0;i1<1990;i1++) {
  char_out = *(pointer);
  fprintf(file_pointer,"%c",char_out);
  i2++;
  if(i2 >= 80) {
    char_out = 0x0D;
    fprintf(file_pointer,"%c",char_out);
    char_out = 0x0A;
    fprintf(file_pointer,"%c",char_out);
    i2 = 0;
  }
  pointer = pointer+2;
}
close_window();
char_out = 0x0D;
fprintf(file_pointer,"%c",char_out);
fclose(file_pointer);
      }
      else {
error_message("File Open Error, Cannot Save!!");
DISK_LOG_UP = FALSE;
      }
      update = TRUE;
    }
  }
}

/*****function to quit , restores interrupt vectors*****/
void finish(void)
{
  char temp;

  if(REMOTE) {
    my_notice("Hanging Up Modem");
    hangup(remote_port);
    release_modem(remote_port);
    close_window(remote_port);
  }
  if(REMOTE_SLAVE) {
    my_notice("Please Wait.  Hanging Up Modem");
    hangup(remote_port);
    set_timer(0,5);
    while(!timed_out(0));
    release_modem(remote_port);
    close_window();
  }
  if(port_up(receiver))
    closeport(receiver);
  if(port_up(remote_port));
```

```
    closeport(remote_port);
  if(PARALLEL_UP)
    closeparallel();
  restore_timer();
  restore_keyboard();
  if(IRIG_UP)
    restore_irig();
  close_all_windows();
  show_cursor();
  clrscr();
//  temp = bioskey(0);
/*****use DOS to exit, C exit() routine call wrong interrupt*****/
  _AH = 0x4C;
  _AL = 0x00;
  geninterrupt(0x21);
}

/*****procedure to capture screens and save to disk
** used to generate screens for thesis report******/
void store_screen(void)
{
  FILE *file_pointer;
  char file_name[11];
  int i1;

  DOS_TIMER_UP = FALSE;
  my_notice("opening file");
  sprintf(file_name,"%c%c%c%c%c%c%c%c%c%c%c",sys_time[0],sys_time[1],
sys_time[2],sys_time[7],sys_time[8],
sys_time[10],sys_time[11],
'.','c','a','p');
  file_pointer = fopen(file_name,"wt");
  if(file_pointer != NULL) {
    close_window();
    my_notice("Saving Screen To Disk ");
    for(i1=0;i1<4096;i1++)
      fprintf(file_pointer,"%c",screen_buffer[i1]);
    close_window();
    fclose(file_pointer);
    my_notice("Any key to continue!!");
    cputs("   FILE = ");
    cputs(file_name);
    i1 = bioskey(0);
    close_window();
  }
  else {
    error_message("File Open Error, Cannot Save!!");
    finish();
  }
}
```

```
/*****************************************************
*File  : menu.h
*By    : Jim Coppola
*      : 17 Aug 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  File contains the header information for the menu
* functions in menu.c.
*
******************************************************/

typedef struct {
  char *mname;            /*menu bar selection names */
  char **mselcs;          /*the pop-down menu selections */
  int (**func)(void);     /*the functions to execute*/
} MENU;

void executive(MENU *);

/*****menu colors*****/
#define MENUBG RED
#define MENUFG WHITE
#define SELECTBG WHITE
#define SELECTFG BLACK
#define DISABLEFG LIGHTGRAY

#define endname(m,h,v) \
  (&(m[h-1].mselcs[v-1][strlen(m[h-1].mselcs[v-1])]))
#define selector(m,h,v) (*(endname(m,h,v)-2))
#define enabler(m,h,v)  (*(endname(m,h,v)-1))

#define CHECK '\xfb'
#define test_option(m,h,v)   (selector(m,h,v)==CHECK)
#define set_option(m,h,v)   (selector(m,h,v)=CHECK)
#define clear_option(m,h,v)   (selector(m,h,v)=' ')
#define enable_selection(m,h,v)  (enabler(m,h,v)=' ')
#define disable_selection(m,h,v) (enabler(m,h,v)='*')
#define enabled(m,h,v)  (enabler(m,h,v)!='*')

/*****************************************************
*File  : menu.c
*By    : Jim Coppola
*      : 17 Aug 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  File contains the menu
```

```
* functions.
*
*   Global Variables Used    : **none**
*   Global Variables Changed : **none**
*******************************************************/
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "window.h"
#include "menu.h"
#include "keys.h"
#include "cursor.h"
#include "system.h"

#define ON 1
#define OFF 0

static int getvmn(MENU *, int*);
static void dimension(char **, int *, int *);
static void light(MENU *, int, int);
static void vlight(MENU *, int, int, int);

static int lastvsel[] = {1,1,1,1,1,1};

/*****display and process a menu******/
void executive(MENU *mn)
{
  int hsel=1, vsel, frtn = FALSE;
  int i = 0;

  my_notice("<ESC> To Exit\r\n<CR> To Select");
  open_window(1,1,80,1,MENUFG,MENUBG,0,0);
  hidecursor();
  cprintf("    ");
  while ((mn+i)->mname)
    cprintf(" %-10.10s ",(mn+i++)->mname);
  light(mn, hsel, ON);
  while (!frtn && ((vsel = getvmn(mn, &hsel)) != 0)) {
    light(mn,hsel,OFF);
    show_cursor();
    frtn = (*(mn+hsel-1)->func[vsel-1])();
    hidecursor();
    light(mn, hsel, ON);
  }
  close_window();
  close_window();
}

/*****pop down a vertical menu*****/
```

```
static int getvmn(MENU *mn, int *hsel)
{
  int ht, wd, vx, sel =0, vsel, vs;
  char **selcs;

  while (sel != ESC && sel != '\r') {
    dimension((mn+*hsel-1)->mselcs, &ht, &wd);
    vx = 5+(*hsel-1)*12;
    open_window(vx, 2, vx+wd+1, ht+3, MENUFG, MENUBG,1,0);
    selcs = (mn+*hsel-1)->mselcs;
    for(vsel = 1; *selcs; vsel++) {
      textcolor(enabled(mn,*hsel,vsel)?MENUFG:DISABLEFG);
      cprintf(vsel == 1 ? " %s" : "\r\n %s",*selcs++);
    }
    textcolor(MENUFG);
    sel = 0;
    vsel = lastvsel[*hsel-1];
    while(sel != ESC && sel != '\r' && sel != FWD && sel != BS &&
  sel != F4) {
      vlight(mn, *hsel, vsel, ON);
      sel = enabled(mn, *hsel, vsel) ?
getkey() :
(sel == UP ? UP : DN);
      selcs = (mn+*hsel-1)->mselcs;
      for(vs = 1; *selcs; vs++, selcs++) {
if(enabled(mn,*hsel,vs) && tolower(**selcs) == tolower(sel)) {
  vsel = vs;
  vlight(mn,*hsel, vsel, ON);
  sel = '\r';
  break;
}
      }
      switch(sel) {
      case UP :
vlight(mn, *hsel, vsel, OFF);
if(vsel > 1)
    vsel;
else
  vsel = ht;
vlight(mn,*hsel, vsel, ON);
      break;
      case DN :
vlight(mn, *hsel, vsel, OFF);
if(vsel < ht)
  vsel++;
else
  vsel = 1;
vlight(mn, *hsel, vsel, ON);
      break;
      case '\r' :
if((mn+*hsel-1)->func [vsel-1] == NULL) {
```

```
  if(test_option(mn,*hsel,vsel))
    clear_option(mn,*hsel,vsel);
  else
    set_option(mn,*hsel,vsel);
  vlight(mn, *hsel, vsel, ON);
  sel = 0;
}
else
  close_window();
      break;
      case FWD :
close_window();
light(mn,*hsel,OFF);
if((mn+*hsel)->mname)
  (*hsel)++;
else
  *hsel = 1;
light(mn,*hsel,ON);
      break;
      case BS :
close_window();
light(mn, *hsel, OFF);
if(*hsel == 1)
  while((mn+*hsel)->mname)
    (*hsel)++;
else
  --(*hsel);
light(mn,*hsel, ON);
break;
case ESC :
  close_window();
break;
default :
  putch(BELL);
break;
      }
    }
  }
  return sel == ESC ? 0 : vsel;
}

/*****compute a menu's height and width*****/
static void dimension(char *sl[], int *ht, int *wd)
{
  *ht = *wd = 0;
  while(sl && sl [*ht]) {
    *wd = max(*wd, (unsigned) strlen(sl [*ht]) + 2);
    (*ht)++;
  }
}
```

```
/*****light a horizontal menu selection*****/
static void light(MENU *mn, int hsel, int onoff)
{
  textcolor(onoff ? SELECTFG : MENUFG);
  textbackground(onoff ? SELECTBG : MENUBG);
  gotoxy((hsel-1)*12+6, 1);
  cprintf((mn+hsel-1)->mname);
}

/*****hightlight a vertical menu selection*****/
static void vlight(MENU *mn, int hsel, int vsel, int onoff)
{
  textcolor(onoff ? SELECTFG : enabled(mn,hsel,vsel) ? MENUFG : DISABLEFG);
  textbackground(onoff ? SELECTBG : MENUBG);
  gotoxy(1,vsel);
  cprintf(" %s", *((mn+hsel-1)->mselcs+vsel-1));
  lastvsel[hsel-1] = vsel;
}
```

```
/***** modem.h *****
 ***** Modem Definitions *****/
/*****Hayes modem control strings *****/
#define RESETMODEM "ATZ/r~"
#define INITMODEM  "AT&C1E0M1s7=60s11=55V1X3s0=0/r~"
#define HANGUP     "~+++~ATH0/r~ATS0=0/r~"
#define ANSWER     "ATS0=1/r"
/***** prototypes *****/
void initializemodem(void);
void call_remote(char *);
void anwser_modem(void);
void hangup(void);
void release_modem(void);


/*******************************************************
*File  : options.c
*By    : Jim Coppola
*      : 19 Aug 92
*      : Version 1.0
*Update:
*By    :
*
*Description: Contains functions to get receiver, get satellite,
* get logging info, and to get system time.
*
*   Global Variables Used    : INITIAL, REMOTE, REMOTE_SLAVE
*   Global Variables Changed : receiver, DOS_TIMER_UP, IRIG_UP,
*B_BIRD, FIRST, sat[], PRINT_TRIGGER, PRINT_LOG_UP, DISK_TRIGGER,
*disk_time, disk_count, print_time, print_count
*******************************************************/

#include <conio.h>
#include <string.h>
#include "system.h"
#include "window.h"
#include "menu.h"

/*****function prototype*****/
void clear_all(void);

/*****menu selections*****/
char *recvselections[] = {
  "GPM        ",
  "Raytheon   ",
  "ASC 30     ",
  "Remote     ",
  NULL
};
```

```
char *timeselections[] = {
  "MS-DOS   ",
  "IRIG     ",
  NULL
};

char *satselections[] = {
  "A1       ",
  "A2       ",
  "A3       ",
  "B4       ",
  "B5       ",
  "B6       ",
  "B7       ",
  "B8       ",
  "B9       ",
  "B10      ",
  "B11      ",
  "B12      ",
  "B13      ",
  "B14      ",
  "Other    ",
  "Remote   ",
  NULL
} ;

char *remselcs[] = {
  "Start Master    ",
  "Start Slave     ",
  "Stop Master     ",
  "Stop Slave      ",
  NULL
} ;

char *logselections[] = {
  "Trigger Print   ",
  "Timed Print     ",
  "Trigger Disk    ",
  "Timed Disk      ",
  NULL
} ;

char *controlselections[] = {
  "GPM Start   ",
  "GPM Window  ",
  "Ray Start   ",
  "Ray Offset  ",
  "Remote      ",
  NULL
} ;
```

```
/*****function declarations for functions called when menu items
are selected*****/
int gp(void),ray(void),a30p(void),rstub(void);
int manual(void),irg(void);
int a1(),a2(),a3(),b4(),b5(),b6(),b7(),b8(),b9(),b10(),b11(),
    b12(),b13(),b14(),other(),remote_stub();
int rm(void),rs(void),krm(void),krs(void);
int tprnt(void),prnt(void),tdsc(void),dsc(void);
int gpmst(void),gpmcont(void),rayst(void),raycont(void);

static int (*recfuncs[])()={gp,ray,a30p,rstub};
static int (*timefuncs[])()={manual,irg};
static int (*satfuncs[])()={a1,a2,a3,b4,b5,b6,b7,b8,b9,b10,b11,
    b12,b13,b14,other,remote_stub};
static int (*remfuncs[])()={rm,rs,krm,krs};
static int (*logfuncs[])()={tprnt,prnt,tdsc,dsc};
static int (*contfuncs[])()={gpmst,gpmcont,rayst,raycont,rstub};

/*****main menu items displayted on top line*****/
static MENU mn[] = {
  {"Receiver",recvselections,recfuncs},
  {"Time",timeselections,timefuncs},
  {"Satellite",satselections,satfuncs},
  {"Remote" ,remselcs,remfuncs},
  {"Log",logselections,logfuncs},
  {"Control",controlselections,contfuncs},
  {NULL}
};

/*****sets default menu options,  kinda a cluge, init should call the
menu functions themselves, update if version two is ever done*****/
void set_default_options(void)
{
  gp();
  set_option(mn,2,1);
  set_option(mn,3,14);
  disable_selection(mn,3,16);
  disable_selection(mn,1,4);
  disable_selection(mn,4,3);
  disable_selection(mn,4,4);
  disable_selection(mn,6,5);
}
/*****call menu software to display these menu data structures*****/
void get_options(void)
{
  executive(mn);
}

/*****set up for gpm receiver*****/
int gp(void)
{
```

```
  receiver = 0;
  enable_selection(mn,6,1);
  enable_selection(mn,6,2);
  disable_selection(mn,6,3);
  disable_selection(mn,6,4);
  clear_option(mn,1,2);
  clear_option(mn,1,3);
  set_option(mn,1,1);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****raytheon receiver selected*****/
int ray(void)
{
  receiver = 1;
  enable_selection(mn,6,3);
  enable_selection(mn,6,4);
  disable_selection(mn,6,1);
  disable_selection(mn,6,2);
  clear_option(mn,1,1);
  clear_option(mn,1,3);
  set_option(mn,1,2);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****using asc-30 as data input device*****/
int a30p(void)
{
  clear_option(mn,1,1);
  clear_option(mn,1,2);
  set_option(mn,1,3);
  receiver = 2;
  if(port_up(beacon_port))
    closeport(beacon_port);
  if(port_up(gpm_control_port))
    closeport(gpm_control_port);
  DOS_TIMER_UP = FALSE;
  IRIG_UP = FALSE;
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****enabled only when remote slave, displays an error message
```

```
to user*****/
int rstub(void)
{
  error_message("Remote Master Controlling Reciever!!");
  return FALSE;
}

/*****start remote master software*****/
int rm(void)
{
  if((receiver==0) || (receiver == 1)) {
    remote_master();
    if(REMOTE){
      set_option(mn,4,1);
      clear_option(mn,4,2);
      disable_selection(mn,4,2);
      disable_selection(mn,4,1);
      enable_selection(mn,4,3);
      enable_selection(mn,6,5);
      disable_selection(mn,6,1);
      disable_selection(mn,6,2);
      disable_selection(mn,6,3);
      disable_selection(mn,6,4);
      return TRUE;
    }
    else
      return FALSE;
  }
  else {
    error_message("Cannot Run Remote With ASC-30 Reciever");
    return FALSE;
  }
}

/*****kill remote master function*****/
int krm(void)
{
  kill_remote_master();
  clear_option(mn,4,1);
  enable_selection(mn,4,2);
  enable_selection(mn,4,1);
  disable_selection(mn,4,3);
  disable_selection(mn,6,5);
  enable_selection(mn,6,1);
  enable_selection(mn,6,2);
  enable_selection(mn,6,3);
  enable_selection(mn,6,4);
  return TRUE;
}

/*****kill_remote slave routine, just adjusts menu selections currently****/
```

```
int krs(void)
{
  int i1;
  finish(); /*****current version just quits.  Stuff after this
  fixes all menu stuff if later versions have someting to do
  before quitting the program *****/
//  kill_remote_slave();
  clear_option(mn,4,2);
  disable_selection(mn,4,4);
  enable_selection(mn,4,1);
  enable_selection(mn,4,2);
  enable_selection(mn,1,1);
  enable_selection(mn,1,2);
  enable_selection(mn,1,3);
  enable_selection(mn,2,2);
  disable_selection(mn,1,4);
  clear_option(mn,1,4);
  set_option(mn,1,1);
  for(i1=1;i1<16;i1++)
    enable_selection(mn,3,i1);
  disable_selection(mn,3,16);
  clear_option(mn,3,16);
  set_option(mn,3,14);
  enable_selection(mn,6,1);
  enable_selection(mn,6,2);
  enable_selection(mn,6,3);
  enable_selection(mn,6,4);
  disable_selection(mn,6,5);
  return FALSE;
}

/*****start remote slave routine*****/
int rs(void)
{
  int i1;

  remote_slave();
  if(REMOTE_SLAVE) {
    disable_selection(mn,4,1);
    disable_selection(mn,4,2);
    disable_selection(mn,4,3);
    enable_selection(mn,4,4);
    set_option(mn,4,2);
    disable_selection(mn,1,1);
    disable_selection(mn,1,2);
    disable_selection(mn,1,3);
    enable_selection(mn,1,4);
    disable_selection(mn,2,2);
    IRIG_UP = FALSE;
    DOS_TIMER_UP = TRUE;
    clear_option(mn,1,1);
```

```
    clear_option(mn,1,2);
    clear_option(mn,1,3);
    set_option(mn,1,4);
    clear_all();
    for(i1=1;i1<16;i1++)
      disable_selection(mn,3,i1);
    enable_selection(mn,3,16);
    set_option(mn,3,16);
    disable_selection(mn,6,1);
    disable_selection(mn,6,2);
    disable_selection(mn,6,3);
    disable_selection(mn,6,4);
    enable_selection(mn,6,5);
    return TRUE;
  }
  else
    return FALSE;
}

/*****dos timer routine*****/
int manual(void)
{

  DOS_TIMER_UP = TRUE;
  if(IRIG_UP)
    restore_irig();
  set_option(mn,2,1);
  clear_option(mn,2,2);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****set system up to use IRIG timer*****/
int irg(void)
{
  int go = TRUE;

  DOS_TIMER_UP = FALSE;
  while (go) {
    my_notice("Initializing IRIG");
    go = start_irig();
    close_window();
    if(!go)
      go = yesno("Try to initialize IRIG again? ");
    else
      go = FALSE;
  }
  if(IRIG_UP) {
    clear_option(mn,2,1);
```

```
    set_option(mn,2,2);
    if(INITIAL)
      return FALSE;
  else
    return TRUE;
  }
  else {
    DOS_TIMER_UP = TRUE;
    return FALSE;
  }
}

/*****clear out all satellite menu selections*****/
void clear_all(void)
{
  int i1;

  for(i1=1;i1<16;i1++)
    clear_option(mn,3,i1);
}

/*****the following set the sat name display to the selected item*****/
int a1()
{
  B_BIRD = FALSE;
  FIRST = TRUE;
  sat[0] = 'A';
  sat[1] = '1';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,1);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int a2()
{
  B_BIRD = FALSE;
  FIRST = TRUE;
  sat[0] = 'A';
  sat[1] = '2';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,2);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}
```

```
int a3()
{
  B_BIRD = FALSE;
  FIRST = TRUE;
  sat[0] = 'A';
  sat[1] = '3';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,3);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b4()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '4';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,4);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b5()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '5';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,5);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b6()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
```

```
  sat[1] = '6';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,6);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b7()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '7';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,7);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b8()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '8';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,8);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b9()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '9';
  sat[2] = ' ';
  clear_all();
  set_option(mn,3,9);
  if(INITIAL)
    return FALSE;
```

```
  else
    return TRUE;
}

int b10()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '0';
  clear_all();
  set_option(mn,3,10);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b11()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '1';
  clear_all();
  set_option(mn,3,11);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b12()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '2';
  clear_all();
  set_option(mn,3,12);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b13()
{
```

```
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '3';
  clear_all();
  set_option(mn,3,13);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

int b14()
{
  B_BIRD = TRUE;
  FIRST = TRUE;
  sat[0] = 'B';
  sat[1] = '1';
  sat[2] = '4';
  clear_all();
  set_option(mn,3,14);
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****get sat name from user.  Assume B model satellite*****/
int other()
{
  char sat_def[6];
  int i1;

  open_window(20,8,60,10,WHITE,GREEN,1,2);
  gotoxy(1,3);
  cputs("Satellite Model? ");
  sat_def[0] = 4;
  my_cgets(sat_def);
  for(i1=2;i1<5;i1++)
    sat[i1-2] = sat_def[i1];
  if(sat[0] == 'A')
    B_BIRD = FALSE;
  else
    B_BIRD = TRUE;
  close_window();
  clear_all();
  set_option(mn,3,15);
  if(INITIAL)
    return FALSE;
  else
```

```
    return TRUE;
}
/****function for menu selection when disabling all others by
     remote_slave function*****/
int remote_stub(void)
{
  error_message("Remote Master Setting Satellite Model!!");
  return FALSE;
}

/*****enable trigger printing*****/
int tprnt(void)
{

  if(test_option(mn,5,1)) {
    clear_option(mn,5,1);
    PRINT_TRIGGER = FALSE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
  else {
    set_option(mn,5,1);
    PRINT_TRIGGER = TRUE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
}

/*****set up timed print*****/
int prnt(void)
{
  char pmin[6], *pointer;
  int temp;

  if(test_option(mn,5,2)) {
    clear_option(mn,5,2);
    PRINT_LOG_UP = FALSE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
  else {
    set_option(mn,5,2);
    open_window(20,8,70,10,WHITE,GREEN,1,2);
    gotoxy(1,3);
    cputs("Minutes Between Printouts? (1 to 120) : ");
```

```
    pmin[0] = 3;
    my_cgets(pmin);
    pointer = pmin + 2;
    temp = atoi(pointer);
    if((temp < 1) || (temp > 120)) {
      error_message("Illegal Log Time!!!!!  Try Again.");
      close_window();
      return FALSE;
    }
    else {
      print_time = (60 * temp)*180/10 + 1;
      print_count = 3*180/10+1;
      PRINT_LOG_UP = TRUE;
      close_window();
      if(INITIAL)
return FALSE;
      else
return TRUE;
    }
  }
}

/*****enable/disable trigger disk.  Function toggles each time when
selected*****/
int tdsc(void)
{

  if(test_option(mn,5,3)) {
    clear_option(mn,5,3);
    DISK_TRIGGER = FALSE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
  else {
    set_option(mn,5,3);
    DISK_TRIGGER = TRUE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
}

/*****enable timed disk saving*****/
dsc(void)
{
  char pmin[6], *pointer;
  int temp;
```

```
  if(test_option(mn,5,4)) {
    clear_option(mn,5,4);
    DISK_LOG_UP = FALSE;
    if(INITIAL)
      return FALSE;
    else
      return TRUE;
  }
  else {
    set_option(mn,5,4);
    open_window(20,8,70,10,WHITE,GREEN,1,2);
    gotoxy(1,3);
    cputs("Minutes Between Disk Saves? (1 to 120) : ");
    pmin[0] = 3;
    my_cgets(pmin);
    pointer = pmin + 2;
    temp = atoi(pointer);
    if((temp < 1) || (temp > 120)) {
      error_message("Illegal Log Time!!!!!  Try Again.");
      close_window();
      return FALSE;
    }
    else {
      disk_time = (60 * temp)*180/10 + 1;
      disk_count = 3*180/10 + 1;
      DISK_LOG_UP = TRUE;
      close_window();
      if(INITIAL)
return FALSE;
      else
return TRUE;
    }
  }
}

/*****gpm start window*****/
int gpmst(void)
{
  int temp;
  temp = gpm_start();
  if(!temp)
    return FALSE;
  if(!INITIAL)
    return TRUE;
  else
    return FALSE;
}

/*****gpm control window*****/
int gpmcont(void)
{
```

```
  gpm_control();
  if(!INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****call ray start software*****/
int rayst(void)
{
  ray_start();
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}

/*****call ray freq offset function*****/
int raycont(void)
{
  ray_control();
  if(INITIAL)
    return FALSE;
  else
    return TRUE;
}
```

```
/* ******************************************************
*File  : pio.h
*By    : Jim Coppola
*      : 25 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Header file for the parallel port function listed in
* pio.c.
*
*
*
****************************************************** */

extern char *nextserialin, *nextserialout;

/*****parallel prototypes*****/

void initport();
int getbit();
void clear_parallel_queue();
void closeparallel();

/* ****serial macros**** */

#define bit_ready() (nextparallelin!=nextparallelout)

/* ****parallel port addresses**** */

/*****parallel port address : set for CPM/P operation 0x310*****/

#define ADDRESS         0x310
#define DATAOUT ADDRESS
#define ADATAIN   ADDRESS
#define BDATAIN         ADDRESS + 1;
#define CONTROL         0x313

/*****parallel interrupt values*****/
#define IRQENABLE       0xF7
#define IRQDISABLE      0x08
#define PARALLELINT     0x0B
#define PIC01 0x21 /*8259 interrupt controller*/
#define PIC00 0x20 /*8259 interrupt controller*/
#define EOI 0x20 /*end of interrupt*/


/*****parallel input interrupt buffer*****/
#define PARALLELBUFFERSIZE 1024
```

```
/*****************************************************************
*    File : pio.c
*    by Jim Coppola
*    15 Jul 92
*
*    File where all 24 bit pio port control functions are kept
*    Functions:
*              initport()
*              getbit()
*              putbit()
*              setportout()
*              setportin()
*
*   Global Variables Used    : **NONE**
*   Global Variables Changed : PARALLEL_UP
*****************************************************************/

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include "pio.h"
#include "timer.h"
#include "system.h"

#define TRUE 1
#define FALSE 0

int PARALLELTIMEOUT = 5;
static char parallel_buffer[PARALLELBUFFERSIZE];
char *nextparallelin = parallel_buffer;
char *nextparallelout = parallel_buffer;
int parallel_count;

static void (interrupt far *oldparallelint)(void);
static void interrupt far newparallelint(void);

/*****clear parallel input buffer*****/
void clear_parallel_queue()
{
  nextparallelin = nextparallelout = parallel_buffer;
  parallel_count = 0;
}


/*****interrupt service routine for parallel input*****/
static void interrupt far newparallelint(void)
{
  int c;

  outp(PIC00,EOI);
```

```
    if(nextparallelin == parallel_buffer + PARALLELBUFFERSIZE)
      nextparallelin = parallel_buffer;
    c = inp(ADATAIN);
    *nextparallelin++ = (char) c;
    parallel_count++;
}

/*****initialize parallel port and set up interrupt vector(s)*****/

void initport()
{
  PARALLELTIMEOUT = 5;

/*****set up interrupt vector*****/

  outp(PIC01, (inp(PIC01) | IRQDISABLE));
  oldparallelint = getvect(PARALLELINT);
  setvect(PARALLELINT, newparallelint);
  outp(PIC01, (inp(PIC01) & IRQENABLE));

  /*configure port*/
  outp(CONTROL,0xB1);
  outp(CONTROL,0x09);

/*****fluch existing parallel interrupts*****/
  inp(ADATAIN);

/*****clear serial buffer*****/
  clear_parallel_queue();
  PARALLEL_UP = TRUE;
}


/*****restore the interrupt vector*****/

void restore_parallelint(void)
{
  if(oldparallelint)
    setvect(PARALLELINT, oldparallelint);
  PARALLEL_UP = FALSE;
}

/*****close the port by resetting vector*****/
void closeparallel()
{
  restore_parallelint();
}

/*****read a character from the input buffer*****/
int getbit()
```

```
{
  int ret;

  set_timer(0,PARALLELTIMEOUT);
  while(!bit_ready()) {
    if(timed_out(parallel_timer))
      return FALSE;
      ;
  }
  if (nextparallelout == parallel_buffer + PARALLELBUFFERSIZE)
    nextparallelout = parallel_buffer;
  --parallel_count;
  ret = *nextparallelout++;
  return (ret);
}

/*****configure portA for output, not used in system*****/
void setportout()
{
  int config = 0xAB;

  outport(CONTROL,config);
}

/*****configure portA for input, used to initialize port.  Probably not
needed, but put it before card was working.*****/
void setportin()
{
  int config = 0x9B;

  outport(CONTROL,config);
}

/*****set a reset command to the command register of the PIO*****/
void resetport()
{
  int config = 0x1B;

  outport(CONTROL,config);
  setportin();
}

/*****write data to the A port.  Not used in system*****/
void putbit(bits)
  int bits;
{
  outport(DATAOUT,bits);
}
```

```
/* ****************************************************
*File  : port.h
*By    : Jim Coppola
*      : 19 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Header file for the serial port function listed in
* port.c.  Code modified from the book extending turbo C professional
* by Al Stevens.
*
*
*
*
*
**************************************************** */

/* ****serial prototypes */

void openport(unsigned,int,int,int,int);
int get_char(unsigned port);
int send_char(unsigned port,unsigned char c);
int xmit_rdy(unsigned port);
int char_rdy(unsigned port);
void clear_serial_queue(unsigned port);
void clear_xmit_queue(unsigned port);
void closeport(unsigned port);

/* ****serial port addresses**** */

/*****8250 UART base port address : COM1 = 3F8, COM2 = 2F8*****/

#define COM1 0x3F8
#define INT1            0x0C
#define COM2            0x2F8
#define INT2            0x0B

#define COM        0x14 /*bios com interrupt*/

/*****serial interrupt values*****/

#define IRQENB1         0xEF
#define DISABLE1        0x10
#define IRQENB2 0xF7
#define DISABLE2        0x08
#define PIC01 0x21 /*8259 interrupt controller*/
#define PIC00 0x20
#define EOI 0x20 /*end of interrupt*/
```

```
/*****line status register values*****/

#define XMIT_DATA_READY 0x20

/*****modem control reguster vlaues*****/

#define DTR 1
#define RTS 2
#define OUT2    8

/*****ASCII serial control characters*****/

#define SOH 1
#define EOT 4
#define ACK 6
#define NAK 0x15
#define CAN 0x18
#define CR      0x0D
/***** interrupt enable register signals*****/

#define DATAREADY 1

/*****serial input interrupt buffer*****/
#define SERIALBUFFERSIZE 1024

/****************************************************
*File  : port.c
*By    : Jim Coppola
*      : 01 Feb 92
*      : Version 1.0
*Update:
*By    :
*
*Description: contains interrupt driven port routines to control
* the RS-232 ports.  COM1 and COM2 ports are supported.  Receive is
* interrupt driven with a 1024 character buffer.  Transmit is polled
* output, it waits until xmit buffer is empty to avoid ovrwrites.
*
*   Global Variables Used    : None
*   Global Variables Changed : None
*
*
****************************************************/
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include "system.h"
#include "port.h"
#include "timer.h"

#define TRUE 1
```

```
#define FALSE 0

/*****set up receive buffers for interrupt service routine*****/
static char buff1[SERIALBUFFERSIZE],buff2[SERIALBUFFERSIZE];
static char xbuff1[SERIALBUFFERSIZE],xbuff2[SERIALBUFFERSIZE];

/*****pointers for queue management*****/
char *nextin1      = buff1;
char *nextout1     = buff1;
char *nextin2      = buff2;
char *nextout2     = buff2;
char *nextxmitin1  = xbuff1;
char *nextxmitout1 = xbuff1;
char *nextxmitin2  = xbuff2;
char *nextxmitout2 = xbuff2;

/*****package variables*****/
int TIMEOUT, MODEMCTL, LINESTATUS, MODEMSTATUS;
int INTIDENT, INTENABLE, RXDATA;

/*****isr declarations*****/
static void (interrupt far *oldint1)(void);
static void (interrupt far *oldint2)(void);
static void interrupt far newint1(void);
static void interrupt far newint2(void);


/*****clear xmit buffer*****/
void clear_xmit_queue(unsigned port)
{
  if(port == 1) {
    nextxmitin1 = nextxmitout1 = xbuff1;
  }
  else {
    nextxmitin2 = nextxmitout2 = xbuff2;
  }
}

/*****clear serial input buffer*****/
void clear_serial_queue(unsigned port)
{
  if(port == 1) {
    nextin1 = nextout1 = buff1;
  }
  else {
    nextin2 = nextout2 = buff2;
  }
}

/*****interrupt service routine for serial input. com1
and com2 supported. This one is for com1*****/
```

```
static void interrupt far newint1(void)
{
  int c;
  outp(PIC00,EOI);
  if(nextin1 == buff1 + SERIALBUFFERSIZE)
    nextin1 = buff1;
  c = inp(COM1);
  *nextin1++ = (char) c;
}

/*****interrupt service routine for serial input com2*****/
static void interrupt far newint2(void)
{
  int c;
  outp(PIC00,EOI);
  if(nextin2 == buff2 + SERIALBUFFERSIZE)
    nextin2 = buff2;
  c = inp(COM2);
  *nextin2++ = (char) c;
}

/*****function to check if port is up *****/
int port_up(unsigned port)
{
  if(port == 1)
    return COM1_UP;
  else
    return COM2_UP;
}

/*****initialize serial port and set up interrupt vector(s)*****/
void openport(port,baud,bits,parity,stop)
  unsigned port;
  int baud, bits, parity, stop;
{
  unsigned char pattern = 0;

  /*****remote slave should have longer timeout*****/
  if(!REMOTE_SLAVE)
    TIMEOUT = 5;
  else
    TIMEOUT = 8;

  switch(baud)
  {
    case 110  : pattern = 0x00;
    break;
    case 150  : pattern = 0x20;
    break;
    case 300  : pattern = 0x40;
    break;
```

```
    case 600  : pattern = 0x50;
    break;
    case 1200 : pattern = 0x80;
    break;
    case 2400 : pattern = 0xA0;
    break;
    case 4800 : pattern = 0xC0;
    break;
    case 9600 : pattern = 0xE0;
    break;
    default   : pattern = 0xA0;
  }

  switch(parity)
  {
    case 0  : pattern = pattern;         /*no parity*/
    break;
    case 1  : pattern = pattern | 0x08;  /*odd parity*/
    break;
    case 2  : pattern = pattern | 0x18;  /*even parity*/
    break;
    default : pattern = pattern;
  }

  switch(stop)
  {
    case 0  : pattern = pattern;     /*no stop bits*/
    break;
    case 1  : pattern = pattern | 0x04;  /*one stop bit*/
    break;
    default : pattern = pattern;
  }

  switch(bits)
  {
    case 7  : pattern = pattern | 0x02;
    break;
    case 8  : pattern = pattern | 0x03;
    break;
    default : pattern = pattern | 0x02;
  }
  /*****use dos interrupt to set port parameters*****/
  _DX = port-1;
  _AH = 0;
  _AL = pattern;
  geninterrupt(COM);

/*****set up interrupt vector*****/
  if(port == 1) {
    MODEMCTL = COM1 + 4;
    INTENABLE = COM1 + 1;
```

```
    RXDATA = COM1;
    INTIDENT = COM1 + 2;
    LINESTATUS = COM1 + 5;
    MODEMSTATUS = COM1 + 6;
    if(oldint1 == NULL)
      oldint1 = getvect(INT1);
    setvect(INT1, newint1);
    outp(MODEMCTL, (inp(MODEMCTL) | DTR | RTS | OUT2));
    outp(PIC01, (inp(PIC01) & IRQENB1));
    outp(INTENABLE, DATAREADY);
    COM1_UP = TRUE;
  }
  else {
    MODEMCTL = COM2 + 4;
    INTENABLE = COM2 + 1;
    RXDATA = COM2;
    INTIDENT = COM2 + 2;
    LINESTATUS = COM2 + 5;
    MODEMSTATUS = COM2 + 6;
    if(oldint2 == NULL)
      oldint2 = getvect(INT2);
    setvect(INT2, newint2);
    outp(MODEMCTL, (inp(MODEMCTL) | DTR | RTS | OUT2));
    outp(PIC01, (inp(PIC01) & IRQENB2));
    outp(INTENABLE, DATAREADY);
    COM2_UP = TRUE;
  }
  outp(PIC00, EOI);

/*****fluch existing srial interrupts*****/
  inp(RXDATA);
  inp(INTIDENT);
  inp(LINESTATUS);
  inp(MODEMSTATUS);

/*****clear serial buffer*****/
  clear_serial_queue(port);
  clear_xmit_queue(port);
}

/*****restore the serial interrupt vector*****/
void restore_serialint(unsigned port)
{
  if(port == 1) {
    if(oldint1)
      setvect(INT1,oldint1);
    outp(PIC01, (inp(PIC01) | DISABLE1));
    COM1_UP = FALSE;
  }
  else {
    if(oldint2)
```

```
      setvect(INT2,oldint2);
    outp(PIC01, (inp(PIC01) | DISABLE2));
    COM2_UP = FALSE;
  }
}

/*****close the serial port by resetting vector*****/
void closeport(unsigned port)
{
  restore_serialint(port);
}

/*****check if port ready for xmit character*****/
int xmit_ready(unsigned port)
{
  unsigned local_port;

  if(port == 1)
    local_port = COM1;
  else
    local_port = COM2;

  if((inp(local_port + 5) & XMIT_DATA_READY) == 0)
    return FALSE;
  else
    return TRUE;
}

/*****check if xmit buffer is empty*****/
int xmit_buffer_empty(unsigned port)
{
  unsigned local_port;

  if(port == 1) {
    if(nextxmitin1 != nextxmitout1)
      return FALSE;
    else
      return TRUE;
  }
  else {
    if(nextxmitin2 != nextxmitout2)
      return FALSE;
    else
      return TRUE;
  }
}


/*****send a char to the serial port*****/
int put_char(unsigned port,unsigned char c)
{
```

```
  unsigned local_port;

  if(port == 1)
    local_port = COM1;
  else
    local_port = COM2;

    /*****wait till port is ready for character*****/
  set_timer(port,TIMEOUT);
  while((inp(local_port + 5) & XMIT_DATA_READY) == 0) {
    if(timed_out(port))
return FALSE;
  }
  outp(local_port, c);
  return TRUE;
}

/*****check for character waiting in input buffer*****/
int input_char_ready(unsigned port)
{
  if(port == 1) {
    if(nextin1 != nextout1)
      return TRUE;
    else
      return FALSE;
  }
  else {
    if(nextin2 != nextout2)
      return TRUE;
    else
      return FALSE;
  }
}

/*****if character in xmit queue, then send it to port for xmit*****/
void xmit_char(unsigned port)
{
  if(port == 1) {
    if(nextxmitin1 != nextxmitout1) {
      if (nextxmitout1 == xbuff1 + SERIALBUFFERSIZE)
nextxmitout1 = xbuff1;
      put_char(port,*nextxmitout1++);
    }
  }
  else {
    if(nextxmitin2 != nextxmitout2) {
      if(nextxmitout2 == xbuff2 + SERIALBUFFERSIZE)
nextxmitout2 = xbuff2;
      put_char(port,*nextxmitout2++);
    }
  }
```

```
}


/*****read a character from the input buffer*****/
int get_char(unsigned port)
{
  unsigned char ret;

  set_timer(port,TIMEOUT);
  while(!input_char_ready(port)) {
    if(REMOTE) {
      if(xmit_ready(remote_port)) {
xmit_char(remote_port);
      }
    }
    if(timed_out(port)) {
      return FALSE;
    }
  }

  if(port == 1) {
    if (nextout1 == buff1 + SERIALBUFFERSIZE)
      nextout1 = buff1;
    ret = *nextout1++;
    if(REMOTE) {
      if((xmit_buffer_empty(remote_port)) && (xmit_ready(remote_port)))
put_char(remote_port, ret);
      else {
if(nextxmitin2 == xbuff2 + SERIALBUFFERSIZE)
  nextxmitin2 = xbuff2;
*nextxmitin2++ = ret;
      }
    }
    return (ret);
  }
  else {
    if (nextout2 == buff2 + SERIALBUFFERSIZE)
      nextout2 = buff2;
    ret = *nextout2++;
    if(REMOTE) {
      if((xmit_buffer_empty(remote_port)) && (xmit_ready(remote_port)))
put_char(remote_port,ret);
      else {
if(nextxmitin1 == xbuff1 + SERIALBUFFERSIZE)
  nextxmitin1 = xbuff1;
*nextin1++ = ret;
      }
    }
    return (ret);
  }
}
```

```
/*****fuction to check and see if a character is ready for calling
routine*****/
int char_rdy(unsigned port)
{
  return input_char_ready(port);
}
```

```
/*****************************************************
*File  : printer.c
*By    : Jim Coppola
*      : 22 Sep 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Functions to control the printer
*
*   Global Variables Used    : None
*   Global Variables Changed : None
*
*****************************************************/

#include <bios.h>
#include "system.h"
#include "interupt.h"

/***** function to check if printer is up and ready *****/
int printer_ready(void)
{
  int status;

  status = biosprint(2,0,print_port);
  if((status & 0x01) || (status & 0x08) || (status & 0x20))
    return FALSE;
  else
    return TRUE;
}

/***** initialize printer  *****/
int initialize_printer(void)
{
  int check;
  check = biosprint(1,0,print_port);
  check++; /*****just for compiler warning *****/
  return (printer_ready());
}

/***** check to see if printer busy *****/
int printer_busy(void)
{
  int status;

  status = biosprint(2,0,print_port);
  status &= 0x80;
  if(status == 0)
    return TRUE;
  else
```

```
    return FALSE;
}

/**** print a character to the system selected printer port *****/
void print_char(char char_in)
{
  int status;

  while(printer_busy());
  status = biosprint(0,char_in,print_port);
  status++; /***just to get rid of compiler warning *****/
}

/**** print a string to the system selected printer port *****/
void printstring(char *string_in)
{
  int status;
  char char_out;

  char_out = *(string_in++);
  while(char_out != 0) {
    print_char(char_out);
    char_out = *(string_in++);
    if(!printer_ready) {
      restore_keyboard();
      error_message("Printer Error. Terminating Print.");
      intercept_keyboard();
    }
  }
}
```

```
/***********************************************************
*   File : ray.c
*   by Jim Coppola
*   18 Jul 92
*   version 1.0
*
*Description:  Contains functions to interface to the GPM beacon port,
*receive the beacon data, then call update routines to decode and display
*the SCT configuration.
*
*   Global Variables Used    : update, restart, data, index, remote,
*                              remote_slave,
*   Global Variables Changed : update, data, index, remote
*
*
*********************************************************/
#include <dos.h>
#include <stdio.h>
#include "system.h"
#include "port.h"

int ray_notice_up;

/*****function to strip bits out of ASCII characters sent by the
Raytheon receiver.  Assume bits packed msb -> lsb 8 at a time*****/
int raybits(int charin)
{
  int i2,temp;
  for(i2=0;i2<8;i2++)
  {
    temp = charin & 0x01;
    if(temp == 0)
      data[index] = 0;
    else
      data[index] = 1;
    charin = charin >> 1;
    index++;
    if(index >= 100)
      return TRUE;
  }
  return FALSE;
}

/*****function to talk to the port driver*****/
int ray_get_char()
{
  unsigned com;
  int char_in, test;

  com = beacon_port;
```

```
  char_in = get_char(com);
  test = timed_out(com);
  if((test) && (!restart)){
    if(!ray_notice_up){
      my_notice("No Data from Beacon Receiver!!!");
      ray_notice_up = TRUE;
      update = FALSE;
    }
    else {
      if(restart) {
close_window();
ray_notice_up = FALSE;
update = TRUE;
      }
    }
  }
  else{
    if(ray_notice_up) {
      close_window();
      ray_notice_up = FALSE;
      update = TRUE;
    }
  }
  return (char_in);
}

/*****function to search for start string sent by Raytheon SABR*****/
void findstart()
{
  char start[5] = {'s','t','a','r','t'};
  int go = TRUE;
  char char_in;
  int i1 = 0;

  while(go)
  {
    if(restart)
      break;
    char_in = ray_get_char();
    if(char_in == start[i1]) {
      i1++;
    }
    else {
      i1 = 0;
    }

    if(i1 >= 5)
      go = FALSE;
  }
}
```

```
/*****main function in ray.c.  Finds starts, gets characters, strips
out beacon bits, then displays the data.  If remote master or slave,
sends/receives the satellite name.*****/
void raytheon()
{
  unsigned com;
  int i1, j1, char_in, go,finished, update_labels;
  char quit[] = "ATHO";
  com = beacon_port;

  /*****if remote slave, rs software opens the com1 port.*****/
  if(!REMOTE_SLAVE)
    openport(com,9600,8,0,1);
  j1 = 0;
  go = 1;

  update_labels = FALSE;
  ray_notice_up = FALSE;

  while(go)
  {
  /*****if restart, get out now!!*****/
    if(restart)
      break;
    if(ray_notice_up) {
      close_window();
      ray_notice_up = FALSE;
      update = TRUE;
    }
    findstart();
    index = 0;
    while(index < 100)
    {
      if(restart)
return;
      char_in = ray_get_char();
      finished = raybits(char_in);
      if(index == 48)
display_clock();
      if(finished)
break;
    }
    display_data();
    if(update_labels){
      display_sat();
      update_labels = FALSE;
    }
    if(REMOTE_SLAVE){
      char_in = 0;
      while(char_in != SOH)
char_in = get_char(remote_port);
```

```
      for(i1=0;i1<3;i1++) {
char_in = get_char(remote_port);
if(char_in != sat[i1]) {
  sat[i1] = char_in;
  update_labels = TRUE;
}
      }
    }
    if(REMOTE) {
      put_char(remote_port,SOH);
      for(i1=0;i1<3;i1++)
put_char(remote_port,sat[i1]);

      if(char_rdy(remote_port)) {
char_in = get_char(remote_port);
if(char_in == quit[j1]) {
  j1++;
  if(j1 >= 3)
    kill_remote_master();
}
else
  j1 = 0;
      }
      else
j1 = 0;
    }
  }
}
```

```
/*****************************************************
*File  : remote.c
*By    : Jim Coppola
*      : 05 Sep 92
*      : Version 1.0
*Update:
*By    :
*
*Description:   Functions to control the remote functions
*
*   Global Variables Used    :
*   Global Variables Changed :  REMOTE, remote_slave
*
*****************************************************/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "timer.h"
#include "system.h"
#include "port.h"

/* ***** Modem Definitions *****/
/*****this code belongs in modem.c, but pulled in for this
application*****/

/*****Hayes modem control strings *****/
#define RESETMODEM "ATZ\r~"
#define INITMODEM  "AT&C1E0M1S7=60S11=55V1X3S0=0\r~"
#define HANGUP     "~+++~ATH0\r~ATS0=0\r~"
#define ANSWER     "ATS0=1\r"

/***** prototypes *****/
void initializemodem(void);
void call_remote(char *);
void anwser_modem(void);
void hangup(void);
void release_modem(void);


char DIAL[] = "ATDT";
char phone_number[25];/*****max phone number length = 25*****/

/***** send a string to the modem *****/
static void modem_out(char *s)
{
  while (*s)  {
    if (*s == '~') {
      set_timer(0,1);
      while(!timed_out(0));
    }
```

```
    else if (!put_char(remote_port,*s))
      break;
    s++;
  }
}

/***** initialize the modem *****/
void initializemodem(void)
{
/*****hardcoded for ZOOM and DATACOM modem*****/
  openport(remote_port,2400,8,0,1);
  modem_out(RESETMODEM);
  modem_out(INITMODEM);
  clear_serial_queue(remote_port);
  clear_xmit_queue(remote_port);
}

/*****release the modem *****/
void release_modem(void)
{
  modem_out(RESETMODEM);
  clear_serial_queue(remote_port);
  clear_xmit_queue(remote_port);
}

/***** place a call *****/
void call_remote(char *phoneno)
{
  modem_out(DIAL);
  modem_out(phoneno);
  modem_out("\r");
  clear_serial_queue(remote_port);
  clear_xmit_queue(remote_port);
}

/***** answer a call *****/
void answer_modem (void)
{
  modem_out(ANSWER);
  clear_serial_queue(remote_port);
  clear_xmit_queue(remote_port);
}

/***** hang up the call *****/
void hangup(void)
{
  modem_out(HANGUP);
  clear_serial_queue(remote_port);
  clear_xmit_queue(remote_port);
}
```

```
/*****function to get phone number to dial*****/
void get_phone_number(void)
{
  char *pointer;

  open_window(20,8,60,10,WHITE,GREEN,1,2);
  gotoxy(1,3);
  cputs("Phone Number To Dial?  ");
  phone_number[0] = 17;
  my_cgets(phone_number);

  close_window();
}

/*****remote master code.  Called from options file*****/
void remote_master(void)
{
  int time_out, go, sync_count, i1;
  char char_in, beacon_out[5], *phone_pointer;

  remote_port = 2;
  my_notice("Initializing Modem");
  initializemodem();
  close_window();
  get_phone_number();
  my_notice("Calling Remote Location");
  phone_pointer = &phone_number[0] + 2;
  call_remote(phone_pointer);
  set_timer(0,5);
  while(!timed_out(0));
  close_window();
  my_notice("Waiting for Remote to Answer");
  set_timer(0,5);
  while(!timed_out(0));

  time_out = 0;
  go = TRUE;

  /*****wait for slave to answer*****/
  while(go) {
    char_in = get_char(remote_port);
    if(timed_out(remote_port)) {
      time_out++;
      if(time_out > 3){
my_notice("Wait Time Out Occured!! Continue??");
char_in = bioskey(0);
close_window();
if(tolower(char_in) == 'y')
  time_out = 0;
else {
  close_window();
```

```
  hangup();
  release_modem();
  return;
}
      }
    }
    else {
      sync_count = 0;
      while(sync_count < 3) {
if(char_in == SOH)
  sync_count++;
else {
  if(!timed_out(remote_port))
    sync_count = 0;
  else
    sync_count++;
}
if(sync_count >= 3)
  go = FALSE;
char_in = get_char(remote_port);
      }
    }
  }
  close_window();

  itoa(receiver,beacon_out,10);
  for(i1=0;i1<3;i1++) {
    put_char(remote_port,SOH);
    put_char(remote_port,beacon_out[0]);
  }

  while(char_rdy(remote_port))
    char_in = get_char(remote_port);

  go = TRUE;
  i1 = 0;
  while(go) {
    if(char_in == ACK) {
      REMOTE = TRUE;
      go = FALSE;
    }
    char_in = get_char(remote_port);
    i1++;
    if(timed_out(remote_port) || (i1 > 3))
      go = FALSE;
  }
  if(!REMOTE) {
    error_message("Handshake Sync Failed, Try Again!!!");
    hangup();
    release_modem();
  }
```

```
  else {
    while(char_rdy(remote_port))
      char_in = get_char(remote_port);
    my_notice("Sync Established");
    set_timer(0,2);
    while(!timed_out(0));
    close_window();
    clear_serial_queue(receiver);
    clear_xmit_queue(remote_port);
  }
}

/*****function to kill remote master functions*****/
void kill_remote_master()
{
  my_notice("Remote Stopped.  Hanging Up Modem");
  hangup();
  release_modem();
  close_window();
  REMOTE = FALSE;
}

/*****function to set up modem and port parameters for remote
slave*****/
void remote_slave(void)
{
  int time_out, go, sync_count, i1;
  char char_in;
  char connect[] = {'C','O','N','N','E','C','T'};

  remote_port = 1;
  my_notice("Initializing Modem");
  initializemodem();
  close_window();
  my_notice("Waiting To Answer Phone.  Any Key To Quit.");
  answer_modem();
  go = TRUE;
  i1 = 0;
  while(go) {
    if(bioskey(1)) {
      close_window();
      my_notice("Clearing Modem");
      hangup();
      release_modem();
      close_window();
      return;
    }
    /*****get characters until modem informs us it connects*****/
    char_in = get_char(remote_port);
    if(char_in == connect[i1])
      i1++;
```

```
    else
      i1 = 0;
    if(i1 >= 7)
      go = FALSE;
  }
  close_window();

  while(char_rdy(remote_port))
    char_in = get_char(remote_port);

    /*****connected, now initiate syncronization*****/
  sync_count = 0;
  go = TRUE;

  while(go) {
    for(i1=0;i1<4;i1++)
      put_char(remote_port,SOH);
    char_in = get_char(remote_port);
    sync_count++;
    if((timed_out(remote_port)) || (sync_count >= 20))
      go = FALSE;
    if(char_in == SOH) {
      char_in = get_char(remote_port);
      if((char_in == '0') || (char_in == '1')) {
REMOTE_SLAVE = TRUE;
for(i1=0;i1<3;i1++)
  put_char(remote_port,ACK);
go = FALSE;
/*****get reciever, either GPM or Raytheon*****/
if(char_in == '0')
  receiver = 0;
else
  receiver = 1;
      }
    }
  }

  while(!char_rdy(remote_port))
    char_in = get_char(remote_port);

  if(!REMOTE_SLAVE) {
    error_message("Handshake Sync Failed, Try Again!!!");
    hangup();
    release_modem();
  }
  else {
    for(i1=0;i1<5;i1++)
      put_char(remote_port,ACK);
    my_notice("Sync Established");
    clear_serial_queue(receiver);
    set_timer(0,2);
```

```
    while(!timed_out(0));
  }
}
```

```
/*****************************************************
*File  : screen.h
*By    : Jim Coppola
*      : 27 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description:  Header file for screen functions
*
*
*****************************************************/

void display_clock(void);
void display_data(void);
void display_system(void):

/*****************************************************
*File  : screen.c
*By    : Jim Coppola
*      : 28 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description: procedures which output to the screen
*
*
*   Global Variables Used    : update, data[], old[], B_BIRD, sys_time,
* mil_date, IRIG_UP, INTERRUPT_UP, FGCOLOR, BGCOLOR, receiver,
* PRINTER_UP, DISK_LOG_UP.
*   Global Variables Changed : old[]
*
*****************************************************/
#include <conio.h>
#include <string.h>
#include "system.h"

/*****global toggles/place holders*****/
int CONTSHF = FALSE;
int BERT    = FALSE;

int OLDSHFPWR  = 0xFF;
int COMMANDPWR = 0xFF;
int BITERRATE  = 0xFF;
int SBITRATE   = 0xFF;
int UPSTRENGTH = 0xFF;
int DEMOD      = 0xFF;

/*****display positions for various blocks and stuff*****/
```

```
int timex = 36;
int timey = 2;
int datex = 70;
int datey = 2;
int clockx = 37;
int clocky = 3;
int diffx = 23;
int diffy = 4;

int statusx = 20;
int statusy = 7;

int configux = 60;
int configuy = 8;
int configdx = 70;
int configdy = 8;

int leftx = 22;
int lefty = 16;
int offset = 19;
int rightx = 65;
int righty = 16;

/*****function prototype****/
void display_labels(void);

/*****routine to update status section of display*****/
void update_status(void)
{
  static char *one1[] = {"1",NULL};
  static char *zero0[] = {"0",NULL};
  char **one, **zero;

  one = one1;
  zero = zero0;
  /*****bypass*****/
  if((data[82]!=old[82]) || (data[83]!=old[83]))
  {
    gotoxy(statusx,statusy);
    CONTSHF = FALSE;
    old[82] = data[82];
    old[83] = data[83];
    if(old[82])
    {
      if(old[83]) {
CONTSHF = TRUE;
cputs("SHF CONT");
      }
      else
cputs("BYPASS  ");
    }
```

```
    else
    {
      if(old[83])
      {
textcolor(FGCOLOR + BLINK);
cputs("  EAM   ");
textcolor(FGCOLOR);
      }
      else
cputs("STANDBY ");
    }
  }

  /*****command accepted*****/
  if(old[72] != data[72]) {
    old[72] = data[72];
    gotoxy(statusx,statusy+1);
    if(old[72]) {
      cputs("ACCEPTED");
      if(PRINT_TRIGGER) {
restart = TRUE;
printnow = TRUE;
      }
    }
    else {
      cputs("RESET   ");
    }
  }
  /*****STG update*****/
  if(data[70]!=old[70])
  {
    gotoxy(statusx,statusy+2);
    old[70] = data[70];
    if(old[70])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }

  /*****serial command*****/
  if((data[75]!=old[75]) || (data[76]!=old[76]) || (data[77]!=old[77]))
  {
    old[75] = data[75];
    old[76] = data[76];
    old[77] = data[77];
    gotoxy(statusx,statusy+3);
    if(old[75])
      cputs(*one);
    else
      cputs(*zero);
    if(old[76])
```

```
      cputs(*one);
    else
      cputs(*zero);
    if(old[77])
      cputs(*one);
    else
      cputs(*zero);
  }

  /*****broadside cmd*****/
  if((data[48]!=old[48]) || (data[49]!=old[49]) || (data[50]!=old[50]))
  {
    old[48] = data[48];
    old[49] = data[49];
    old[50] = data[50];
    gotoxy(statusx,statusy+4);
    if(old[48])
      cputs(*one);
    else
      cputs(*zero);
    if(old[49])
      cputs(*one);
    else
      cputs(*zero);
    if(old[50])
      cputs(*one);
    else
      cputs(*zero);
    cputs("  ");
  }
  '*****decover*****/
  if(data[80]!=old[80])
  {
    gotoxy(statusx,statusy+5);
    old[80] = data[80];
    if(old[80])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }
}

/*****routine to update the config part display******/
void update_config(void)
{

  /*****uplink band*****/
  if((data[68]!=old[68]) | (data[69] != old[69]))
  {
    gotoxy(configux,configuy);
    old[68] = data[68];
```

```
    old[69] = data[69];
    if(old[68]) {
      if(old[69]) {
textcolor(FGCOLOR+BLINK);
cputs("COMMUTATE");
textcolor(FGCOLOR);
      }
      else
cputs("UHF      ");
    }
    else
      cputs("SHF      ");
  }

  /*****uplink bandwidth*****/
  if(data[58] != old[58]) {
    old[58] = data[58];
    gotoxy(configux,configuy+1);
    if(old[58])
      cputs("NARROW");
    else
      cputs("WIDE  ");
  }

  /*****uplink modulation*****/
  if(data[56] != old[56]) {
    old[56] = data[56];
    gotoxy(configux,configuy+2);
    if(old[56])
      cputs("AFSAT II");
    else
      cputs("AFSAT I ");
  }

  /*****uplink crypto*****/
  if(data[78] != old[78]) {
    old[78] = data[78];
    gotoxy(configux,configuy+3);
    if(old[78])
      cputs("ILSG");
    else
      cputs("KI  ");
  }

  /*****uplink AFSAT II status*****/
  if(old[74] != data[74]) {
    old[74] = data[74];
    gotoxy(configux,configuy+4);
    if(old[56]) {
      if(old[74])
cputs("Hopping");
```

```
      else
cputs("Fixed  ");
    }
    else
      cputs("       ");
  }

  /*****update downlink side of configuration menu*****/
  /*****downlink band*****/
  if(B_BIRD) {
    if((data[90]!=old[90]) | (data[91] != old[91])) {
      gotoxy(configdx,configdy);
      if(data[90]) {
if(data[91])
  cputs("UHF/SHF");
else
  cputs("UHF    ");
      }
      else  {
if(data[91])
  cputs("SHF    ");
else
  cputs("NONE   ");
      }
    }
  }
  else {
      gotoxy(configdx,configdy);
      cputs("UHF    ");
  }

  /*****downlink bandwidth*****/
  if(data[59] != old[59]) {
    old[59] = data[59];
    gotoxy(configdx,configdy+1);
    if(old[59])
      cputs("WIDE  ");
    else
      cputs("NARROW");
  }

  /**** downlink modulation*****/
  if(data[57] != old[57]) {
    old[57] = data[57];
    gotoxy(configdx,configdy+2);
    if(old[57])
      cputs("AFSAT I ");
    else
      cputs("AFSAT II");
  }
  /*****downlink crypto*****/
```

```
  if(data[79] != old[79]) {
    old[79] = data[79];
    gotoxy(configdx,configdy+3);
    if(old[79])
      cputs("LSG ");
    else
      cputs("KI  ");
  }

  /*****downlink AFSAT II status*****/
  if(old[73] != data[73]) {
    old[73] = data[73];
    gotoxy(configdx,configdy+4);
    if(old[57])
      cputs("       ");
    else {
      if(old[73])
cputs("Hopping");
      else
cputs("Fixed  ");
    }
  }
}

/*****update left block of beacon display for B sat*****/
void update_leftb(void)
{
  int i1,value;
  char cvalue[5];

  /*****afsatcom I bypass enable*****/
  if(data[81] != old[81]) {
    old[81] = data[81];
    gotoxy(leftx,lefty);
    if(old[81])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }

  /*****Command address enable*****/
  if(data[52] != old[52]) {
    old[52] = data[52];
    gotoxy(leftx,lefty+1);
    if(old[52])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }
  /*****Uplink frequency enable*****/
  if(data[53] != old[53]) {
```

```
  old[53] = data[53];
  gotoxy(leftx,lefty+2);
  if(old[53])
    cputs("ENABLED ");
  else
    cputs("DISABLED");
}

/*****BERT*****/
if(BERT) {
  value = 0;
  for(i1=0;i1<7;i1++) {
    value = value << 1;
    value |= data[98-i1];
  }
  if(value != BITERRATE) {
    BITERRATE = value;
    gotoxy(leftx,lefty+3);
    itoa(value,cvalue,10);
    cputs(cvalue);
    cputs("        ");
  }
}
else {
    gotoxy(leftx,lefty+3);
    cputs("DISABLED");
}

/*****commanded shf EAM dnlink power*****/
value = 0;
for(i1=0;i1<6;i1++) {
  value = value << 1;
  value |= data[89-i1];
}
if((value != COMMANDPWR || FIRST)) {
  COMMANDPWR = value;
  gotoxy(leftx-offset,lefty+4);
  cputs("Commanded SHF Pwr ");
  gotoxy(leftx,lefty+4);
  itoa(value,cvalue,10);
  cputs(cvalue);
  cputs("        ");
}

/*****Uplink strength or cont SHF downlink power*****/
if(data[67]) {
  if(data[67] != old[67]) {
    old[67] = data[67];
    gotoxy(leftx-offset,lefty+5);
    cputs("Cont SHF Dnlk    ");
    gotoxy(leftx-offset,lefty+6);
```

```
        cputs("SHF Dnlk Pwr     ");
      }
      if(old[66] != data[66]) {
        old[66] = data[66];
        gotoxy(leftx,lefty+5);
        if(data[66])
cputs("ENABLED ");
        else
cputs("DISABLED");
      }
      value  = 0;
      for(i1=0;i1<6;i1++) {
        value = value << 1;
        value |= data[65-i1];
      }
      if((value != OLDSHFPWR) || (FIRST)) {
        gotoxy(leftx,lefty+6);
        OLDSHFPWR = value;
        itoa(value,cvalue,10);
        cputs(cvalue);
        cputs("        ");
      }
    }
    else {
      gotoxy(leftx-offset,lefty+5);
      cputs("Uplink Strength  ");
      gotoxy(leftx-offset,lefty+6);
      cputs("Demod Data       ");
      if(BERT) {
        value = 0;
        for(i1=0;i1<4;i1++) {
value = value << 1;
value |= data[60+i1];
        }
        if(value != UPSTRENGTH) {
UPSTRENGTH = value;
gotoxy(leftx,lefty+5);
itoa(value,cvalue,10);
cputs(cvalue);
cputs("       ");
        }
        value = 0;
        for(i1=0;i1<3;i1++) {
value = value << 1;
value |= data[66-i1];
        }
        if(value != DEMOD) {
gotoxy(leftx,lefty+6);
itoa(value,cvalue,10);
cputs(cvalue);
cputs("       ");
```

```
      }
    }
    else {
      gotoxy(leftx,lefty+5);
      cputs("DISABLED");
      gotoxy(leftx,lefty+6);
      cputs("DISABLED");
    }
  }
}

/*****update left block of beacon display for A sat*****/
void update_lefta(void)
{
  int i1,value ;
  char cvalue[5];

  /*****afsatcom I bypass enable*****/
  if(data[81] != old[81]) {
    old[81] = data[81];
    gotoxy(leftx,lefty);
    if(old[81])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }

  /*****Command address enable*****/
  if(data[52] != old[52]) {
    old[52] = data[52];
    gotoxy(leftx,lefty+1);
    if(old[52])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }
  /*****Uplink frequency enable*****/
  if(data[53] != old[53]) {
    old[53] = data[53];
    gotoxy(leftx,lefty+2);
    if(old[53])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }

  /*****BERT*****/
  if(BERT) {
    value = 0;
    for(i1=0;i1<7;i1++) {
      value = value << 1;
```

```
      value |= data[98-i1];
    }
    if(value != BITERRATE) {
      BITERRATE = value;
      gotoxy(leftx,lefty+3);
      itoa(value,cvalue,10);
      cputs(cvalue);
      cputs("       ");
    }
    value = 0;
    for(i1=0;i1<8;i1++) {
      value = value << 1;
      value |= data[91-i1];
    }
    if(value != SBITRATE) {
      SBITRATE = value;
      gotoxy(leftx,lefty+4);
      itoa(value,cvalue,10);
      cputs(cvalue);
      cputs("       ");
    }
  }
  else {
      gotoxy(leftx,lefty+3);
      cputs("DISABLED");
      gotoxy(leftx,lefty+4);
      cputs("DISABLED");
  }

  /*****Uplink strength, DATA DEMOD*****/
  if(BERT) {
    value = 0;
    for(i1=0;i1<4;i1++) {
      value = value << 1;
      value |= data[60+i1];
    }
    if(value != UPSTRENGTH) {
      UPSTRENGTH = value;
      gotoxy(leftx,lefty+5);
      itoa(value,cvalue,10);
      cputs(cvalue);
      cputs("       ");
    }
    value = 0;
    for(i1=0;i1<3;i1++) {
      value = value << 1;
      value |= data[66-i1];
    }
    if(value != DEMOD) {
      gotoxy(leftx,lefty+6);
      itoa(value,cvalue,10);
```

```
      cputs(cvalue);
      cputs("        ");
    }
  }
  else {
  gotoxy(leftx,lefty+5);
  cputs("DISABLED");
  gotoxy(leftx,lefty+6);
  cputs("LISABLED");
  }
}

/*****routine to update right block*****/
void update_right()
{
  /*****UHF downlink - bit 91*****/
  if(B_BIRD) {
    if(old[90] != data[90]) {
      old[90] = data[90];
      gotoxy(rightx,righty);
      if(old[90])
cputs("ENABLED ");
      else
cputs("DISABLED");
    }
  }
  else {
    gotoxy(rightx,righty);
    cputs("**N/A** ");
  }

  /*****SHF downlink enable = bit 92*****/
  if(B_BIRD) {
    if(old[91] != data[91]) {
      old[91] = data[91];
      gotoxy(rightx,righty+1);
      if(old[91])
cputs("ENABLED ");
      else
cputs("DISABLED");
    }
  }
  else {
    gotoxy(rightx,righty+1);
    cputs("**N/A** ");
  }

  /*****SHF antenna - bit 72*****/
  if(old[71] != data[71]) {
    old[71] = data[71];
    gotoxy(rightx,righty+2);
```

```
    if(old[71])
      cputs("EC ");
    else
      cputs("MBA");
  }
  /*****WOD1 - bit 55*****/
  if(old[54] != data[54]) {
    old[54] = data[54];
    gotoxy(rightx,righty+3);
    if(old[54])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }

  /*****WOD2 - bit 56*****/
  if(old[55] != data[55]) {
    old[55] = data[55];
    gotoxy(rightx,righty+4);
    if(old[55])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }
  /*****classified telemetry*****/
  if(old[99] != data[99]) {
    old[99] = data[99];
    gotoxy(rightx,righty+5);
    if(old[99])
      cputs("ENABLED ");
    else
      cputs("DISABLED");
  }
  /*****AFSAT I demod data for A satellites*****/
  if(!B_BIRD) {
    if(old[67] != data[67]) {
      old[67] = data[67];
      gotoxy(rightx, righty+6);
      if(old[67])
cputs("Mark ");
      else
cputs("Space");
    }
  }
}

/*****function to display time using global time character array*****/
void display_time(void)
{
  int x,y;
```

```
  if(update) {
    x = wherex();
    y = wherey();
    gotoxy(timex,timey);
    cputs(sys_time);
    cputs(".000        ");
    gotoxy(datex,datey);
    cputs(mil_date);
    gotoxy(x,y);
  }
}

/*****generate and display beacon clock time*****/
void display_clock()
{

  int day,hour,min,sec,msec,usec,i1;
  char cday[3],chour[3],cmin[3],csec[3],cmsec[7],cdiff[3];
  static int value[7] = {40,20,10,8,4,2,1};
  static int mvalue[7] = {800,400,200,100,80,40,20};

  day=hour=min=sec=msec=0;
  if(update) {
    for(i1=0;i1<7;i1++)
    {
      day=day + data[i1] * value[i1];
      if(i1>0)
hour=hour + data[i1+6] * value[i1];
      min = min + data[i1+13] * value[i1];
      sec = sec + data[i1+20] * value[i1];
      msec = msec + data[i1+27] * mvalue[i1];
    }
    if(IRIG_UP && INTERRUPT_UP) {
      i1 = diff_time - sec;
      if(i1 < 0)
i1 += 60;
      itoa(i1,cdiff,10);
      gotoxy(diffx,diffy);
      cputs("Difference              ");
      cputs(cdiff);
      cputs("     ");
    }

    msec = msec + data[34] * 10;
    msec = msec + data[35] * 5;

    gotoxy(clockx,clocky);
    itoa(day,cday,10);
    itoa(hour,chour,10);
    itoa(min,cmin,10);
    itoa(sec,csec,10);
```

```
    itoa(msec,cmsec,10);
    if(day < 10)
      cputs(" ");
    cputs(cday);
    cputs("/");
    if(hour < 10)
      cputs(" ");
    cputs(chour);
    cputs(" ");
    if(min < 10)
      cputs("0");
    cputs(cmin);
    cputs(" ");
    if(sec < 10)
      cputs("0");
    cputs(csec);
    cputs(".");
    cputs(cmsec);
    cputs("  ");
  }
}

/**** routine to display current satellite *****/
void display_sat()
{
  int i1;

  for(i1=0;i1<100;i1++)
    old[i1] = 0xFF;
  FIRST = TRUE;
  if(sat[0] == 'A')
    B_BIRD = FALSE;
  else
    B_BIRD = TRUE;
  display_labels();
}

/*****routine to update screen labels*****/
void display_labels()
{
  int i1;

  gotoxy(1,1);
  textcolor(RED);
  textbackground(LIGHTGRAY);
  cputs("                              DSCSIII ");
  cputs(sat);
  cputs(" Beacon Display                                    ");

  textcolor(FGCOLOR);
  textbackground(BGCOLOR);
```

```
  gotoxy(23,2);
  cputs("System Time");

  gotoxy(23,3);
  cputs("Beacon Time");

  gotoxy(1,6);
  cputs("  Status                                    Configuration \r\n");
  cputs("  Downlink                                                Uplink    Downlink  \r\n");
  cputs("  Command Accept                            Band \r\n");
  cputs("  STG Update                                Bandwidth \r\n");
  cputs("  Serial Cmd                                Modulation \r\n");
  cputs("  Broadside Cmd                             Crypto \r\n");
  cputs("  Decover                                   AFSAT II");

  gotoxy(1,16);
  cputs("  AFSAT I Bypass                            UHF Downlink    \r\n");
  cputs("  Command Address                           SHF Downlink   \r\n");
  cputs("  Uplink Frequency                          SHF Antenna  \r\n");
  cputs("  Bit Error Rate                            WOD 1 \r\n");
  if(B_BIRD){
    cputs("                                            WOD 2\r\n");
    cputs("                                            Classified Tlmtry");
  }
  else {
    cputs("  Bit Error Rate                            WOD 2 \r\n");
    cputs("  Uplink Strength                           Classified Tlmtry\r\n");
    cputs("  Demod Data                                AFSAT I Baseband");
  }

  gotoxy(1,24);
  textcolor(RED);
  textbackground(LIGHTGRAY);
  cputs("  Quit  Options  Print  Receiver Satellite IRIG  Print log  Disk log          ");

  for(i1=2;i1<24;i1++)
  {
    gotoxy(1,i1);
    cputs(" ");
    gotoxy(80,i1);
    cputs(" ");
  }
  textcolor(FGCOLOR);
  textbackground(BGCOLOR);
}

/*****function to  update the status line*****/
void display_system(void)
{
  gotoxy(1,25);
```

```
  cputs("  <Q>    <F1>    <F2>  ");

  switch (receiver) {
    case 0 : cputs("   GPM    ");
  break;
    case 1 : cputs(" Raytheon ");
  break;
    case 2 : cputs("  ASC-30  ");
  break;
    default : cputs("  GPM     ");
  }

  if(!B_BIRD) {
    cputs("DSCSIII A  ");
  }
  else
    cputs("DSCSIII B  ");

  if(IRIG_UP)
    cputs("Up   ");
  else
    cputs("NA   ");

  if(PRINTER_UP) {
    if(PRINT_LOG_UP)
      cputs("Logging    ");
    else
      cputs("Ready      ");
  }
  else
    cputs("  Down     ");

  if(DISK_LOG_UP)
    cputs("Logging");
  else
    cputs("Off    ");

}

/*****main routine, used to update bits 49-100.  Called from receiver
software.  Clock called separately.*****/
void display_data()
{
  hidecursor();
  if(data[51])
    BERT = TRUE;
  else
    BERT = FALSE;
  if(update) {
    update_status();
    update_config();
```

```
    if(B_BIRD)
      update_leftb();
    else
      update_lefta();
    update_right();
  if(FIRST)
    FIRST = FALSE;
  }
}
```

```c
/*****************************************************
*File  : stime.c
*By    : Jim Coppola
*      : 15 Sep 92
*      : Version 1.0
*Update:
*By    :
*
*Description:   Functions to implement system timer through MS-DOS
*  and display
*   Global Variables Used    :  IRIG_UP
*   Global Variables Changed : sys_time, mil_date
*
******************************************************/

#include <dos.h>
#include "system.h"

/****global variables*****/
char dos_time[14];
unsigned char old_time;
int old_jdate;

/*****calendar conversion routine.  Used to convert MS-DOS mil date
format to linear day of year.  Jan 1 is considered day 1*****/
float datetojulian(int day, int month, int year)
{
  float d, m, y, c, ya, ret, temp, t1, t2, t3, t4;

  if(month > 2)
    month = month-3;
  else {
    month = month + 9;
    year = year - 1;
  }

  d = (float) day;
  m = (float) month;
  y = (float) year;
  temp = y/100;
  c = (long) (temp);
  ya = (long) (y - 100.0 * c);
  t1 = (long) ((146097.0 * c)/4);
  t2 = (long) ((1461*ya)/4);
  t3 = (153*m+2)/5;
  t4 = (long) (t3 + d+1721119.0);
  ret = (long) (t1 + t2 + t4);
  if(IRIG_UP)
    irig_first_day = ret;
  return (ret);
```

```
}

/*****function to generate date when required*****/
void init_dos()
{
  struct date dos_date;
  int i1, jdate, month_offset;
  char temp[5];
  float julian_date, first_day;
  char months[] = {"JanFebMarAprMayJunJulAugSepOctNovDec"};

  getdate(&dos_date);
  julian_date = datetojulian(dos_date.da_day,dos_date.da_mon,
     dos_date.da_year);
  first_day = datetojulian(1,1,dos_date.da_year);
  julian_date = julian_date - first_day + 1;
  jdate = (int) julian_date;
  if(jdate != old_jdate) {
    old_jdate = jdate;
    itoa(dos_date.da_day,temp,10);
    if(dos_date.da_day < 10) {
      mil_date[0] = '0';
      mil_date[1] = temp[0];
    }
    else {
      mil_date[0] = temp[0];
      mil_date[1] = temp[1];
    }
    mil_date[2] = ' ';
    month_offset = (dos_date.da_mon - 1) *3;

    for(i1=0;i1<3;i1++)
      mil_date[i1+3] = months[i1 + month_offset];
    mil_date[6] = ' ';
    itoa(dos_date.da_year,temp,10);
    mil_date[7] = temp[2];
    mil_date[8] = temp[3];
    mil_date[9] = 0; /**null terminator**/
  }
  itoa(jdate,temp,10);
  for(i1=0;i1<14;i1++)
    dos_time[i1] = ' ';
  dos_time[3] = '/';
  if(jdate < 10) {
    dos_time[2] = temp[0];
    return;
  }
  if(jdate < 100) {
    dos_time[1] = temp[0];
    dos_time[2] = temp[1];
    return;
```

```
  }
  for(i1=0;i1<3;i1++)
    dos_time[i1] = temp[i1];
}

/****function called by timer interrupt when 1 second up*****/
/*****keep it short as possible!!!!!*****/
void put_time(void)
{
  struct time dostime;
  char *pointer = dos_time + 4;
  int i1;

  gettime(&dostime);
  sprintf(pointer,"%02d %02d %02d",dostime.ti_hour, dostime.ti_min,
    dostime.ti_sec);
  if(dostime.ti_hour != old_time)  {
    old_time = dostime.ti_hour;
    init_dos();
  }
  for(i1=0;i1<14;i1++)
    sys_time[i1] = dos_time[i1];
  display_time();
}
```

```
/*
*   File: SYSTEM.H
*
*   by Jim Coppola
*   1 Feb 92
*   version 1.0
*
*/

#ifdef MAIN
#define MAINEXTERN
#else
#define MAINEXTERN extern
#endif

#define PIO 0x310  /*parallel port address */
#define TRUE 1
#define FALSE 0
#define SYS_UP TRUE
#define DOWN FALSE
#define SPACE 0x20
#define LF 0x0A

void finish(void);

MAINEXTERN int index, old[100], data[100];
MAINEXTERN int FGCOLOR, BGCOLOR;
MAINEXTERN int receiver,B_BIRD,FIRST, INITIAL;
MAINEXTERN int IRIG_UP,COM1_UP,COM2_UP,PARALLEL_UP, PRINTER_UP;
MAINEXTERN int PRINTER_UP, DISK_LOG_UP;
MAINEXTERN unsigned com1_timer, com2_timer, parallel_timer;
MAINEXTERN unsigned beacon_port, gpm_control_port, print_port;
MAINEXTERN int quitnow, options, update, restart, printnow, disknow;
MAINEXTERN int REMOTE, REMOTE_SLAVE, remote_port, START_REMOTE;
MAINEXTERN char sat[3], mil_date[10], sys_time[14];
MAINEXTERN unsigned dos_timer_count;
MAINEXTERN int DOS_TIMER_UP, INTERRUPT_UP;
MAINEXTERN float irig_first_day;
MAINEXTERN unsigned long print_time, print_count, disk_time, disk_count;
MAINEXTERN int PRINT_LOG_UP, PRINT_TRIGGER, DISK_TRIGGER;
MAINEXTERN int diff_time, capturenow;
MAINEXTERN char screen_buffer[4096];
```

```
/* ****************************************************
*File   : timer.h
*By     : Jim Coppola
*       : 19 Jul 92
*       : Version 1.0
*Update:
*By     :
*
*Description: Header file for the timer functions.  Taken from extending
* turbo C professional by Al Stevens
*
*
******************************************************/

/* ****timer prototypes**** */

void intercept_timer(void (*)(void));
void restore_timer(void);
void set_timer(unsigned timer, int secs);
int timed_out(unsigned timer);

#define PIC00 0x20
#define EOI     0x20

/* ****************************************************
*File   : timer.c
*By     : Jim Coppola
*       : 19 Jul 92
*       : Version 1.0
*Update: 10 Aug 92
*By     : Jim Coppola
*
*Description: File for the timer functions.  Taken from extending
* turbo C professional by Al Stevens. Added in multiple timer
* capability for specific functions.
*
*   Global Variables Used    : DISK_LOG_UP, DOS_TIME_UP, disk_time,
* print_time, PRINT_LOG_UP,
*   Global Variables Changed : disk_count, print_count, com1_timer,
* com2_timer, parallel_timer, dos_time_count, PRINT_NOW
*
***************************************************** */

#include <stdio.h>
#include <dos.h>
#include "system.h"
#include "timer.h"
#include "interupt.h"
```

```
/*****function prototypes*****/
static void (interrupt far *oldtimer)(void);
static void interrupt far newtimer(void);
static void (*timeout_function)(void);

/*****intercept the timer interrupt vector*****/
void intercept_timer(void (*fn)(void))
{
//  if(oldtimer == NULL)
//  {
    oldtimer = getvect(TIMER);
    setvect(TIMER, newtimer);
//  }
  timeout_function = fn;
}

/*****restore timer interrupt vector*****/
void restore_timer()
{
//  if(oldtimer != NULL)
    setvect(TIMER, oldtimer);
  timeout_function = NULL;
}

/*****ISR to count timer ticks*****/
static void interrupt far newtimer()
{
  (*oldtimer)();

  if(com1_timer > 0)
    if(--com1_timer == 0 && timeout_function != NULL)
      (*timeout_function)();

  if(com2_timer > 0)
    com2_timer--;

  if(parallel_timer > 0)
    parallel_timer--;

  if(DOS_TIMER_UP) {
    dos_timer_count--;
    if(dos_timer_count == 0) {
      dos_timer_count = 17;
      put_time();
    }
  }
  if(PRINT_LOG_UP) {
    print_count--;
    if(print_count == 0) {
      print_count = print_time;
      restart = TRUE;
```

```
      printnow = TRUE;
    }
  }
  if(DISK_LOG_UP) {
    disk_count--;
    if(disk_count == 0) {
      disk_count = disk_time;
      restart = TRUE;
      disknow = TRUE;
    }
  }
}

/*****set a timer for timeout watching*****/
void set_timer(unsigned timer, int secs)
{
  int ltime;

  ltime = secs*182/10+1;
  switch(timer) {
    case 0 : parallel_timer = ltime;
    break;
    case 1 : com1_timer = ltime;
    break;
    case 2 : com2_timer = ltime;
    break;
    default :  ;
  }
}

/*****check to see if particular timer has expired*****/
int timed_out(unsigned timer)
{
  int ret;

  ret = FALSE;

  switch(timer) {
    case 0 : if(parallel_timer == 0)
       ret = TRUE;
    break;
    case 1 : if(com1_timer == 0)
       ret = TRUE;
    break;
    case 2 : if(com2_timer == 0)
       ret = TRUE;
    break;
    default : ;
  }
  if(restart)
    return TRUE;
```

```
  else
    return (ret);
}
```

```
/*****************************************************
*File  : window.h
*By    : Jim Coppola
*      : 28 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description: Header file for pop-up window routines.  Routines taken
* from "Extending Trubo C professional by Al Stevens.
*
*
*****************************************************/

void open_window(int,int,int,int,int,int,int,int);
void close_window(void);
void close_all_windows(void);
void clear_window(void);
void error_message(char *);
void notice(char *);
void my_notice(char *);
int yesno(char *);

#define MAX_WINDOWS 10
#define TRUE 1
#define FALSE 0

/*****************************************************
*File  : window.c
*By    : Jim Coppola
*      : 28 Jul 92
*      : Version 1.0
*Update:
*By    :
*
*Description: File for pop-up window routines.  Routines taken
* from "Extending Trubo C professional by Al Stevens.
*
*   Global Variables Used    :  None
*   Global Variables Changed :  None
*
*****************************************************/

/*****include files*****/

#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <bios.h>
#include <ctype.h>
```

```
#include "window.h"
#include "keys.h"

/*****window definition structures*****/
struct text_info windows [MAX_WINDOWS];
struct text_info wkw;  /*working window*/

/*****window dressing*****/
struct
{
  int frame; /*true if window has a frame*/
  int shadow; /*0=no,1=transparent, 2=opaque*/
  char *wsave; /*points to video memory save buffer*/
} dressing [MAX_WINDOWS];
int curr_wnd = 1; /*current window*/

static int post_message(char *, int, int, int);

/*****open a new window*****/
void open_window(left,top,right,bottom,foreg,backg,frame,shadow)
{
  int bfsize;
  int sinc = 0;

  if(shadow && right < 80 && bottom < 25)
    sinc = 1;
  bfsize = (bottom-top+1+sinc) * (right-left+1+sinc) * 2;
  if(curr_wnd < MAX_WINDOWS)
  {
    if(curr_wnd == 1)
      gettextinfo(windows);
    else
    {
      windows[curr_wnd-1].curx = wherex();
      windows[curr_wnd-1].cury = wherey();
    }
    if((dressing[curr_wnd].wsave=malloc(bfsize))!=NULL)
    {
      gettext(left,top,right+sinc,bottom+sinc,
  dressing[curr_wnd].wsave);
      window(left,top,right,bottom);
      textcolor(foreg);
      textbackground(backg);
      gettextinfo(&wkw);
      dressing[curr_wnd].frame = frame;
      dressing[curr_wnd].shadow = shadow;
      clear_window();
      windows[curr_wnd++] = wkw;
    }
  }
}
```

```
/*****window frame characters*****/
#define NW   (dressing[curr_wnd].frame == 1 ? '\xda' : '\xc9')
#define NE   (dressing[curr_wnd].frame == 1 ? '\xbf' : '\xbb')
#define SE   (dressing[curr_wnd].frame == 1 ? '\xd9' : '\xbc')
#define SW   (dressing[curr_wnd].frame == 1 ? '\xc0' : '\xc8')
#define SIDE (dressing[curr_wnd].frame == 1 ? '\xb3' : '\xba')
#define LINE (dressing[curr_wnd].frame == 1 ? '\xc4' : '\xcd')

/*****blank the window and draw its frame*****/
void clear_window(void)
{
  int x,y;
  int ht = wkw.winbottom - wkw.wintop + 1;
  int wd = wkw.winright - wkw.winleft + 1;
  char line[81];

  clrscr();
  if(dressing[curr_wnd].shadow)
  {
    textcolor(LIGHTGRAY);
    textbackground(BLACK);
    window(wkw.winleft,wkw.wintop,wkw.winright+1,
 wkw.winbottom+2);
    for(y=2;y<ht+1;y++)
    {
      gotoxy(wd+1,y);
      putch(dressing[curr_wnd].shadow == 2 ? ' ' :
  *(dressing[curr_wnd].wsave+((wd+1)*y-1)*2));
    }
    gotoxy(2,ht+1);
    for(x=0;x<wd;x++)
      putch(dressing[curr_wnd].shadow == 2 ? ' ' :
   *(dressing[curr_wnd].wsave+(((wd+1)*ht+1)+x)*2));
    window(wkw.winleft,wkw.wintop,wkw.winright,wkw.winbottom);
    textattr(wkw.attribute);
  }
  if(dressing[curr_wnd].frame)
  {
    window(wkw.winleft,wkw.wintop,wkw.winright,wkw.winbottom+1);
    memset(line+1,LINE,wd-2);
    line[0] = NW;
    line[wd-1] = NE;
    line[wd] = '\0';
    cputs(line);
    for(y=2;y<ht;y++)
    {
      gotoxy(1,y);
      putch(SIDE);
      gotoxy(wd,y);
      putch(SIDE);
```

```
    }
    line[0] = SW;
    line[wd-1] = SE;
    cputs(line);
    window(wkw.winleft+1,wkw.wintop+1,wkw.winright-1,wkw.winbottom-1);
    wkw.curx = wkw.cury = 1;
    gotoxy(1,1);
  }
}

/*****close a window*****/
void close_window(void)
{
  int sinc = 0;

  if(dressing[curr_wnd-1].shadow)
    sinc = 1;
  if(curr_wnd > 1) {
    puttext(wkw.winleft,wkw.wintop,wkw.winright+sinc,
 wkw.winbottom+sinc,dressing[curr_wnd-1].wsave);
    free(dressing[curr_wnd-1].wsave);
    wkw = windows[(--curr_wnd)-1];
    textattr(wkw.attribute);
    if(dressing[curr_wnd-1].frame)
      window(wkw.winleft+1,wkw.wintop+1,
   wkw.winright-1,wkw.winbottom-1);
    else
      window(wkw.winleft,wkw.wintop,wkw.winright,wkw.winbottom);
    gotoxy(wkw.curx,wkw.cury);
  }
}

/*****close all windows to clear screen*****/
void close_all_windows(void)
{
  while(curr_wnd > 1)
    close_window();
}

/*****write an error message*****/
void error_message(char *ermsg)
{
  post_message(ermsg, YELLOW, RED, FALSE);
}

/*****write a notice*****/
void notice(char *note)
{
  post_message(note, WHITE, GREEN, FALSE);
}
```

```
/*****my post message returns without getting a keyboard input*****/
static void my_post_message(char *s,int foreg, int backg)
{
  int c;
  int lf = (80-strlen(s)+2)/2;
  int rt = lf+max(strlen(s)+2,15);
  open_window(lf,11,rt,14,foreg,backg,1,2);
  cputs(s);
  if(backg == RED)
    putch(BELL);
}

/*****my notice returns without getting a keyboard input*****/
void my_notice(char *note)
{
  my_post_message(note, WHITE, GREEN);
}

/*****ask a question*****/
int yesno(char *ques)
{
  return post_message(ques, BLACK, WHITE, TRUE);
}

/*****post a message*****/
static int post_message(char *s,int foreg, int backg, int test)
{
  int c;
  int lf = (80-strlen(s)+2)/2;
  int rt = lf+max(strlen(s)+2,15);
  open_window(lf,11,rt,14,foreg,backg,1,2);
  cputs(s);
  if(backg == RED)
    putch(BELL);
  cputs(test ? "\r\n (Y/N) ..." : "\r\n any key ...");
  do
  {
    c = bioskey(0) & 255;
  } while(test && tolower(c) != 'y' && tolower(c) != 'n');
  close_window();
  return(test && tolower(c) == 'y');
}
```

References

1. J. D. Coppola, "AN/ASC-30 DSCS III SHF stand alone beacon receiver," June 1990. Technical Memorandum, WRDC-TM-90-12, Wright Research and Development Center, Avionics Laboratory, Wright-Patterson AFB Ohio.
2. D. H. Martin, *Communications Satellites 1958 - 1988*. El Segundo, California: The Aerospace Corporation, Dec 1986.
3. J. D. Coppola, "Memo on SCT beacon parameters," June 1991.
4. J. J. Foshee, "User commanding of the single channel transponder," in *IEEE 1988 National Aerospace and Electronics Conference - NAECON 1988*, pp. 1027–1032, May 1988.
5. J. D. Coppola, "System time generator adjustment on the single channel transponder," June 1989. Technical Memorandum, WRDC-TM-89-12, Wright Research and Development Center, Avionics Laboratory, Wright-Patterson AFB Ohio.
6. D. Laux, "SCT telemetry display (mod 2)," 1986. Technical Memorandum, WRDC-TM-86-17, Wrighht Research and Development Center, Avionics Laboratory, Wright-Patterson AFB Ohio.
7. D. Robinson, "Using bus level time code processors to synchronize multiple signal processing stations," in *ISE '89. Eleventh Annual Ideas in Science and Electronics Exposition and Symposium*, pp. 165–170, May 1989.
8. P. Lacy, "Trends in microwave measurements," in *Southcon/87 Conference Record*, vol. 7, Mar 1987.
9. M. C. Chimene, "Architectural considerations for a generic multi-port digital interface," in *ITC/USA '90*, vol. 26, pp. 631–634, International Telemetering Conference, Oct 1990.
10. D.A. Simms and H. J. Butterfield, "PC-based PCM telemetry data reduction system hardware," in *Proceedings of the Thirty-Sixth International Instrumentation Symposium*, vol. 36, pp. 275–283, May 1990.
11. P. Zimmerman, "Toolkit for automated test software development," in *Proceedings -ATE Central*, pp. 26–41, Oct. 1985.
12. I. C. Source, "Model dio-24 reference manual," 1989. Industrial Computer Source Product Manual.
13. BANCOMM Corporation, "bc630at real time clock module, operation and technical manual," October 1991. Version 3.1.
14. CommQuest Technologies, Inc., "CQM-248 digital psk satellite modem," July 1992. Installation and Operating Manual.
15. J. Rumbaugh et al, *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.
16. A. Stevens, *Extending Turbo C Professional*. Portland Oregon: Management Information Source, Inc., 1989.
17. D. D. McLeod, *Turbo Building Blocks*. Greesboro, North Carolina: COMPUTE! Publications, Inc., 1987.
18. T. Hogan, *The Programmer's PC Sourcebook*. Redmond, Washington: Microsoft Press, 1988.
19. Science Applications International Corporation, "Single channel transponder system (SCTS) operator's handbook," January 1992. CDRL Sequence No.: A007C-04 DCS/Communications and Computers Headquarters Strategic Air Command, Offutt Air Force Base, Nebraska.

Vita

Captain James Dee Coppola was born on 22 February, 1959 in Tucson Arizona. He graduated from Buena High School, Sierra Vista, Arizona in 1977. He then attended Central Arizona College where he earned an Associates in Science degree in 1979. After Central Arizona, he attended Northern Arizona University. In 1984 he graduated with a Bachelor of Science in Electrical Engineering Technology. He then enlisted in the Air Force and attended Officers Training School at Lackland AFB and was commissioned a Second Lieutenant on 07 August, 1984. Following OTS, Lt Coppola was assigned to Louisiana Tech University in the Undergraduate Education and Conversion Program where he earned a Bachelor of Science degree in Electrical Engineering in 1986. He was then assigned to Wright Laboratory at Wright-Patterson AFB, Ohio where he worked in the Information Processing Group for two years. In 1988 Captain Coppola was assigned to the Satellite Communications Group in Wright Laboratory where he was the Flight Test Director for the airborne satellite communications testbed. In May 1991, he began a Master's Degree program at the AFIT School of Engineering, Wright-Patterson AFB, Ohio.

Permanent address: 7845 North 1st Avenue
Tucson, Az 85718

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
December 1992

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
SATCOM GENERAL PURPOSE MODEM
DSCS III SCT
BEACON TELEMETRY DISPLAY

5. FUNDING NUMBERS

6. AUTHOR(S)
James D. Coppola

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/92D-13

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
James J. Foshee
WL/AAAI-1
WPAFB, OH 45433-6543

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis documents the design and implementation of a DSCS III Single Channel Transponder (SCT) beacon telemetry display. The system is a personal computer based design which interfaces to three SCT beacon receiver/demodulators. The software was designed to decode and display both the DSCS III A and DSCS III B satellite beacons. Recordings of the SCT beacon display can be made on paper and/or magnetic media when triggered by the user, a watchdog timer, or the SCT command accept telemetry bit. In addition, the system can be configured with an IRIG B Universal Time Coordinates (UTC) card which enables the software to determine the difference between the decoded SCT clock time and the local IRIG time source. Remoting the SCT configuration display is also possible using Hayes-compatible modems over a telephone link.

14. SUBJECT TERMS
DSCS III Beacon, Satellite Telemetry Display, Single Channel Transponder

15. NUMBER OF PAGES
201

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102